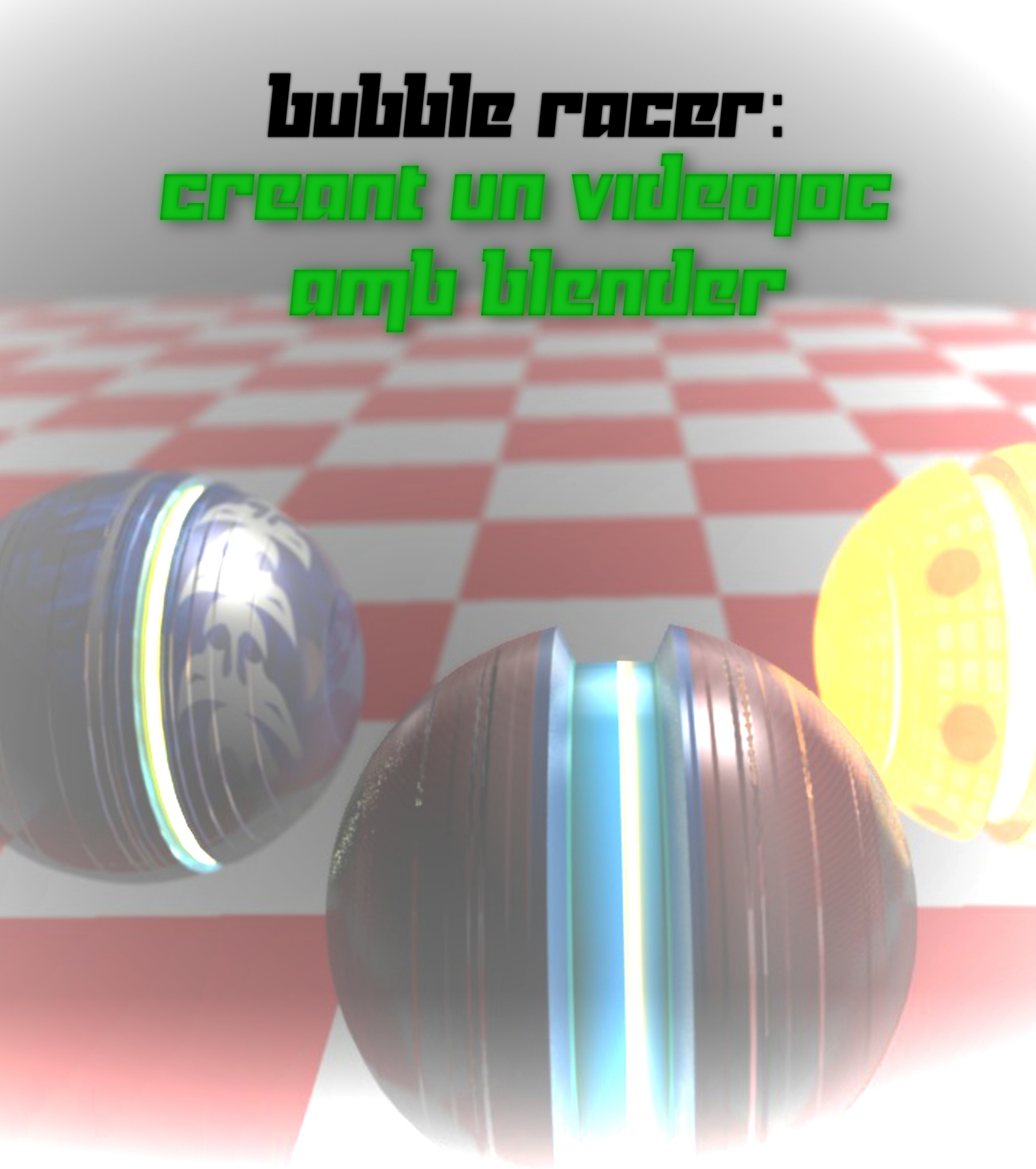
bubble racer:
creant un videojoc amb blender



Robert Planas Jimenez

2n BAT A
Tutor: Martí Vilarnau
INS Narcís Monturiol

INDEX

1. Introducció

1.1. Motivacions

L'industria de l'entreteniment és una de les més poderoses del món. Els videojocs estan influint enormement en la vida de les generacions actuals fins el punt que és difícil trobar adolescents que no en facin ús. Un dels somnis de molts d'aquests jugadors és crear algun dia el seu propi videojoc. Jo comparteixo aquest somni des de sempre i no és la primera vegada que intent-ho materialitzar-lo en una realitat. Aleshores jo encara m'estava iniciant en el món del ordenadors, cap als 13 anys vaig crear el meu primer videojoc. Era un joc molt senzill creat amb un *game engine* anomenat *GameMaker 7.0*. La meva ambició em va portar a intentar crear un joc més complex, en 3D. El primer que vaig fer va ser decantar-me per *Entidad 3D*, un *game engine* que prometia molt, però al poc temps em vaig adonar que aquesta eina em limitava les possibilitats, em privava de llibertat. Només podies fer jocs de dispars i a mi sempre m'havien agradat més els de conducció. Uns anys més tard vaig provar *Blitz3D*, haig de dir al seu favor que no et limitava les possibilitats, però la seva precària *interface*, la falta de documentació, i la seva condició de software llicenciat sota les restriccions de copyright, em va portar a abandonar-lo.

Totes aquestes experiències i desil·lusions m'han servit per aprendre que sempre és possible millorar i crear millors projectes. En aquest treball m'he proposat crear un videojoc partint d'eines avançades amb les que rés sigui impossible, i a la vegada gratuïtes perquè qualsevol pugui seguir els meus passos.

1.2. Pla del treball

No he seguit un pla de treball fix, sinó que he anat buscant informació i creant el videojoc en el meu temps lliure, quan m'ha vingut de gust. Això però, no significa que hi hagi dedicat poc temps, dons utilitzar aquest mètode m'ha permès aprendre un llenguatge de programació nou que no coneixia, Python. Al mateix temps que m'ha permès aprendre les bases del modelatge i la texturització en Blender.

A finals de 2011 vaig començar a experimentar, vaig estar dubtant entre dos versions de Blender, la antiga versió 2.49b i la nova 2.61, fins que finalment em vaig decantar per la versió més nova, molt més moderna gràficament que la anterior. A finals de Maig ja tenia a punt alguns aspectes importants del script de moviment i de la pista. Un cop començat l'estiu els progressos en el joc es varen disparar tant com el temps que hi dedicava. I així en poques setmanes ja havia fet gran part de la feina de modelatge. Ara faltava escriure tot el dossier i començar a fons amb la programació en Python, que és essencialment, la base d'aquest treball.

Pel que fa la documentació escrita o informe del treball, no m'és possible resumir cada concepte del treball detalladament, o cada part del codi font, ja que requeriria superar el nombre de pàgines permeses o per contra, presentar un concentrat extens i il·legible. És per això que només explicaré els conceptes més importants vinculats directament amb el treball, saltant-me aspectes com la programació de la web, la configuració de la xarxa, la creació de les textures, i alguns altres. I explicaré exclusivament temes que tinguin a veure amb el modelatge i la programació.

1.3. Llibertat

Una de les parts més importants del treball és que tot el programari del que es faci ús serà *freeware*, és a dir, totalment lliure. Ara bé, no només el programari, també els recursos del videojoc que inclou textures, músiques i fonts s'ha procurat que fossin amb llicencia gratuïta i per a ús comercial. Per tant, no hi hauria cap impediment legal per a la comercialització de Bubble Racer, encara que segurament aquest també acabara sent *freeware* i de codi obert.

1.4. Comentaris

La recerca d'informació, pel que fa a la llibreria BGE de Python, ha estat realment complicada, molt més del que em pensava als inicis. Tot i aixi les webs de documentació de Blender juntament amb les de Python m'han ajudat molt. M'ha costat trobar altres, necessàries, fonts de informació. Però finalment a data d'avui n'he pogut compilar-ne un bon grapat, que podeu veure, juntament amb algunes webs importants, a la webgrafia.

2. El software

2.1. El motor de joc

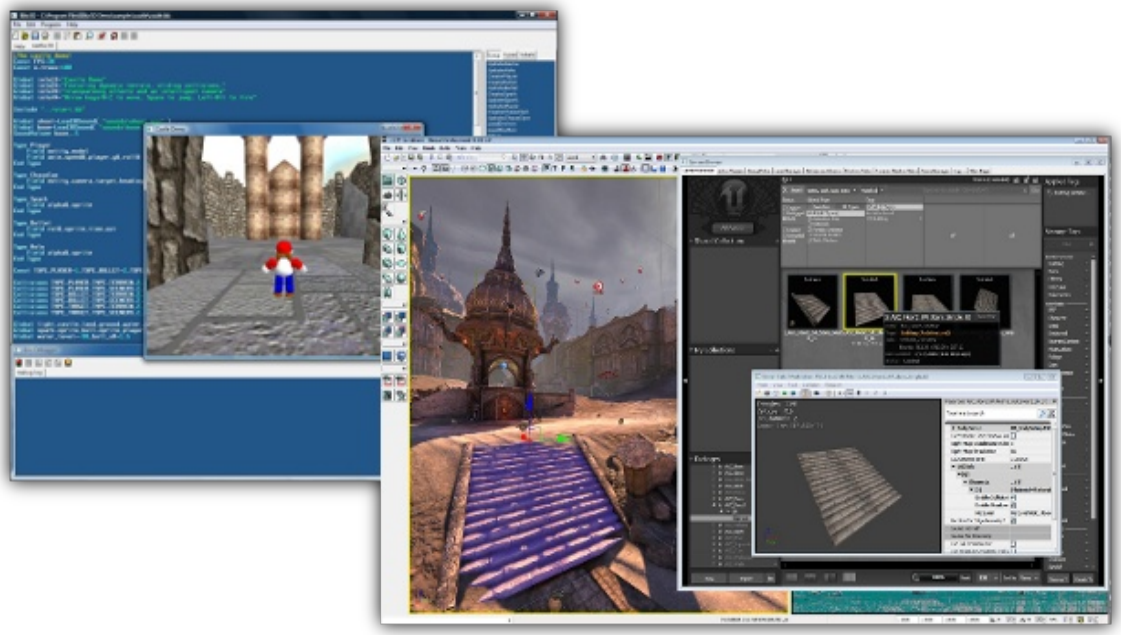
Motor de joc o *game engine* és tot aquell programari destinat a la creació d'un videojoc indiferentment de la seva complexitat o funcionament. Existeix una amplia varietat de motors de joc, cada un diferent. Entre les coses més importants que hauríem de tenir en compte en un motor de joc hi ha: els scripts, la plataforma, la documentació i la llicencia. Entenem per a script un codi que ens permet programar el joc. No tots els motors de joc en tenen ja que alguns estan tan limitats que només permeten programar el joc des de la seva *interface*. Per altra banda no tots els motors de joc tenen *interface*. A continuació podem veure en una taula les característiques dels principals motors de joc disponibles.

Nom	**Llicencia**	**Script**	**Comunitat**	**Plataforma**
Blender	Freeware (GPL)	Python	Reduida[1]	OpenGL (+GLSL)
Unity	Limitat	C++/Java/Python	Important	Windows/Mac/PlayStation/ Xbox/Android/JRE
Blitz3D	Pagament	BASIC	Reduïda	Windows
Game Maker	Limitat	GM Script (Propi)	Mitjana	Windows i Mac
Entidad 3D	Freeware	----------	Mitjana	Windows
FPS Creator	Limitat	FPSC Script (Propi)	Mitjana	Windows
Panda3D	Freeware (BSD)	C++/Python	Mitjana	Llibreria
Macromedia Flash	Pagament	Action Script	Important	Flash Player (Web)
Cry Engine 3	Limitat	C++	Important	La gran majoria.
Unreal Game Engine	Pagament	C++	Important	La gran majoria.
JMonkeyEngine	Freeware (BSD)	Java	Mitjana	JRE

Podeu trobar més informació en: http://en.wikipedia.org/wiki/List_of_game_engines

1 La comunitat de Blender és molt important, ara bé, pel que fa el Game Engine resulta molt escassa.

Una llicencia *freeware* ens indica que el software és gratuït en totes les seves funcions. En canvi una llicencia limitada permet descarregar una versió reduïda del programa, amb les funcions més importants no disponibles, i t'obliga a comprar una llicencia comercial si vols comercialitzar el teu videojoc.



En els scripts hem de tenir en compte quin és el llenguatge de programació que s'utilitzara. I també si aquest serà compilat o interpretat. Un llenguatge compilat és més ràpid de processar ja que no necessites un programa extern per fer-lo funcionar, sinó que és el propi sistema operatiu (SO) el que el processa. En canvi un llenguatge interpretat requereix una altre programa que el processi. Python, Java i ActionScript són llenguatges interpretats.

Entenem per comunitat el conjunt d'usuaris que utilitzen el mateix software i que en parlen en un portal, blog o foro especialitzat.

Per últim, considerem plataforma, no el SO en el que funcionara el *game engine*, sinó el programa o SO que s'encarrega de processar el joc un cop compilat. En el cas de llenguatges interpretats aquest serà JRE, Flash Player o l'interpret de Python. En el cas de llenguatges compilats serà directament el SO (Windows, Linux, PlayStation, Symbian, Android, etc...)

En el cas de Blender la plataforma és qualsevol sistema que suporti OpenGL i Python, molt recomanablement també ha de permetre GLSL.

2.2. Obtenció de les eines

Per la creació de Bubble Racer s'ha usat Blender 2.6x, però sempre és recomanable descarregar l'última versió del programa. El pots trobar a la seva pàgina oficial: http://www.blender.org/

Un altre programa important que també s'ha fet servir és The GIMP, un programa de edició de imatges *freeware*.

2.3. Blender

Blender és un programa que ha evolucionat enormement en els últims anys. El canvi més visible és sens dubte la modernització de la seva *Interface* a partir de Blender 2.5. Blender no és només un *game engine*, de fet, aquesta és només una de les seves funcions. Blender són totes les eines necessitaries, pel que fa al software, per produir una pel·lícula o un videojoc. Exemple d'això n'és el projecte *Sintel*[2] creat completament amb Blender i software lliure. Blender també te altres grans projectes que podem consultar a:

http://www.blender.org/features-gallery/movies/

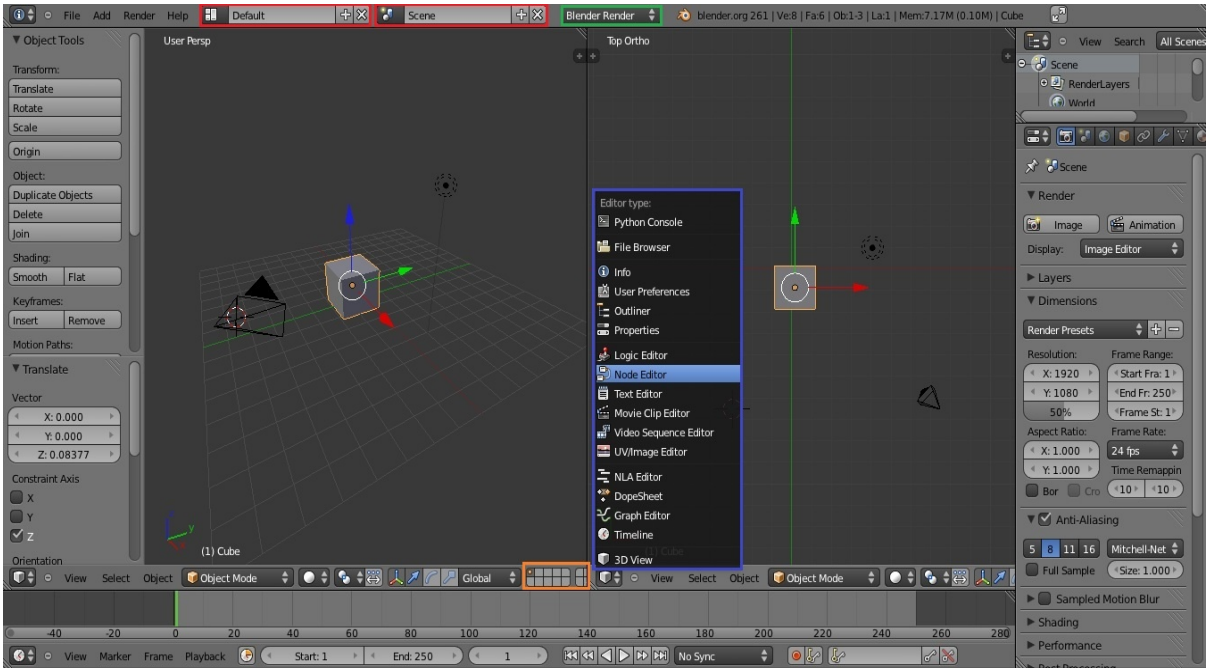
Val a dir que com la majoria del software *freeware* és àmpliament personalitzable. Blender oculta opcions en llocs remots. I si tot i la gran quantitat d'opcions que te, no existeix la que tu necessites, sempre tens la possibilitat de crear-la de zero tu mateix gràcies a la seva condició de codi obert.

La organització de Blender es basa en panells, configuracions de vistes, escenes i capes. En els panells s'agrupen les diverses opcions i eines. Hi ha un panell de opcions, un de configuracions, un de scripts, un de *game engine*, un editor de vídeo, un *timeline*, un de vista 3D, un de tractament de imatges, etc.

2 Projecte Sintel: http://www.sintel.org; http://www.youtube.com/watch?v=eRsGyueVLvQ&hd=1

Pots organitzar els panells *(blau)* com prefereixis, però per no estar reorganitzant-los constantment, ja que tots junts no hi caben a la pantalla. Tenim un llista de configuracions de panells *(vermell)* la qual podem ampliar amb les nostres pròpies configuracions, veure *figura 1*. Tot el que fem a Blender quedara guardat en una escena *(vermell)*. Podem tenir tantes escenes com vulguem. Això ens serà molt útil a l'hora de crear nivells o menús per al joc. Podem moure un objecte entre escenes usant les tecles “Ctrl+L”.

Per últim tenim las capes *(taronja)*. Cada escena consta de 20 capes. Les capes funcionen de forma similar a les escenes, però en pots activar varies al mateix temps. Per canviar un objecte de capa usem la tecla “M”.



2.4. Manuals d'iniciació

Crec convenient facilitar algunes fonts per iniciar-se en Blender i comprendre la complexitat i potencia d'aquest programa. A continuació tenim una guia de la antiga versió 2.4 en català.

http://punttic.cat/files/MaterialsFormatius/26/manual_blender.pdf

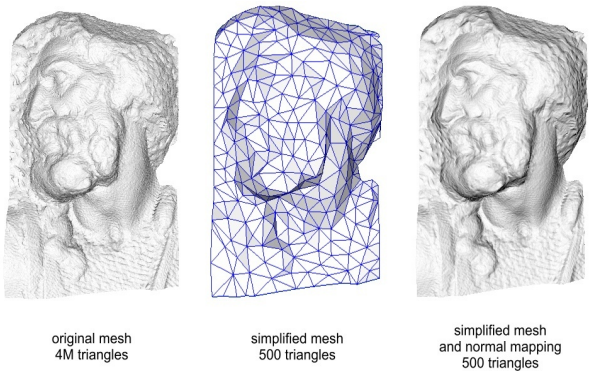
No he trobat cap manual complet, menys en català, sobre la creació de videojocs en Blender, per això he agut de buscar informació en altres llocs, sovint en Anglès. Un lloc que sí

recomano enormement és el foro i blog: http://www.blendernation.com/, un lloc de trobada per aquells interessats en el món 3D, especialment en Blender.

3. Models 3D

Abans de crear qualsevol model tenim que pensar en el rendiment. Com més complex sigui el model més li costarà a l'ordenador de processar el joc. Això ens impedeix utilitzar tècniques de modelatge de molts polígons, com el mode esculpir, i per tant programes especialitzats com Zbrush no compliran bé aquesta funció. Els models, senzills s'anomenen *low-poly*, de pocs polígons. Podem trobar models a Internet, a pàgines com la següent, però en aquest treball crearem els models de zero.

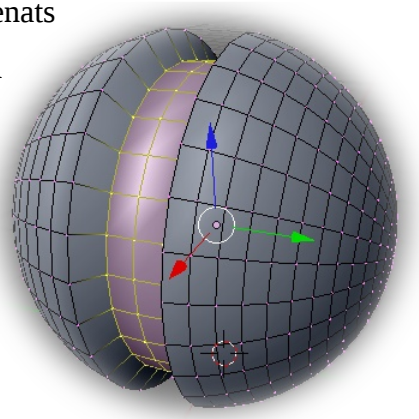
http://www.turbosquid.com/



Un model massa senzill pot resultar poc realista, però això com a alternativa s'usen *normal maps*. Un mapa de normals és una imatge o una textura que dona profunditat als objectes segons el seu valor de blanc o negre. Un exemple d'això ho podeu veure a la imatge superior.

3.1. Creació de les *Bubbles*

En Blender els models són creats a partir d'objectes anomenats "primitives". Per crear la nostre *bubble* usarem una primitiva del tipus *UVsphere* (esfera). Per poder modificar aquesta esfera al nostre gust tenim que entrar en el mode edició, podem prémer la tecla "tabulador" per això. Un cop en el mode edició podem editar directament vèrtex, eixos, o cares del objecte.



Podem seleccionar tots els vèrtex usant la tecla “A”, si la tornem a púlsar es de-seleccionen. Com que volem girar la malla i no l'objecte, per tal de no modificar les coordenades locals, seleccionem tots els vèrtex i girem l'objecte 90 graus en l'eix X. “R*(Rotate)*; X; 90”.

Ara crearem una pertorbació al mig de la malla. Amb la selecció de vèrtex activada, pressionem la tecla “ALT” i a continuació un eix. Tots els eixos consecutius es marcaran formant un anell. Podem seleccionar altres anells amb “SHIFT+ALT” i aixi obtenir anells de cares. En aquest cas seleccionem les dos cares més pròximes al eix paral·lel del mig. Ara ja només falta extrudir-les “E”*(Extrude)*. Extrudir ens serveix per crear una altre conjunt de vèrtex igual, enllaçat a la malla. Sense aquesta eina hauríem de copiar els vèrtex manualment amb “SHIFT+D”*(Duplicate)*, destruir la cara sobrant “X”, i omplir les cares que ens interessin “F”*(Face)*, una feina bastant pesada.

Al moment de extrudir no en modifiquem la posició, sinó que simplement cliquem “ENTER” per confirmar l'extrusió. A continuació pressionem “S”*(Scale)*. Escalar vol dir que tots els vèrtex seleccionats s'ajuntaran o separaran per tal d'augmentar o reduir la grandària de la selecció. Igual que hem fet amb la rotació, si ara escrivim un numero podrem operar amb precisió. Aixi dons escalem en tots els eixos a 1:1,25, o el que és el mateix 0,8. Cal tenir en compte que en l'escriptura anglesa i en programació, les comes que marquen decimals s'escriuen com a punts, i les comes que marquen els milers no s'escriuen, així el numero “2.100,67” s'escriu com “2100.67”. Podem acabar d'ajustar el gruix si tornem a escalar però aquest cop només en l'eix Y: “S;Y;1.1”

Ara només fa falta donar-l'hi color a la *bubble*. Per fer-ho anem al menú de materials del panell d'opcions i afegim un nou material. Aquí hi podrem especificar propietats que facin que tingui un aspecte de plàstic, de ferro, de goma, etc. Ara anem al menú de textures i afegim una nova textura del tipus *Image or Move*. Seleccionem la textura que haurem creat prèviament en GIMP. A continuació anem a la pestanya *Mapping* del menú de textures i en el botó de coordenades marquem UV. De moment no tenim cap UV Map creat, per crear-ne un de manera ràpida anem a la configuració de finestres i seleccionem UV Editing. Un cop allà ens situem al panell de vista 3D i en el menú View seleccionem Orthograpyc i Right. “Numpad 5; Numpad 3”. A continuació amb tots els vèrtex seleccionats anem a Mesh > UV Unwrap >

Project From View. Podem veure'n els resultats si activem el mode *Texture* del panell 3D. Si no ha quedat com volem, podem ajustar la projecció en el panell d'imatge, seleccionant la imatge sobre la qual volem treballar.

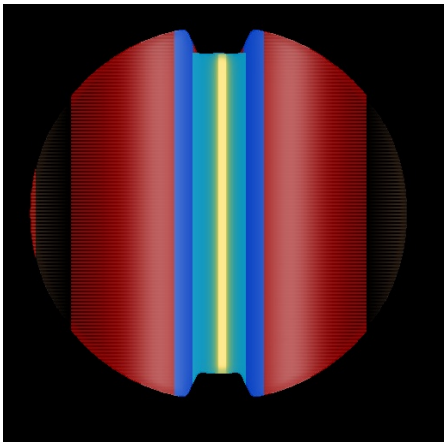


Fig 2. Textura de color de la Bubble 1

Fig 3.Textura de normals de la Bubble 1

Per aplicar la textura de normals ho tenim que fer igual que amb la textura de color, però a més tenim que marcar la casella *Normal* de la pestanya *Influence* del menú de textures. Per últim podem suavitzar les cares marcant el boto *Smooth* de les eines ràpides de la vista 3D.

3.2. Creació de la pista

La pista com a tal, no és més que un pla amb una textura de grava. A sobre d'aquest pla simple hi trobem un objecte que és el recorregut de la pista; quatre objectes de *checkpoint*, i alguns objectes que defineixen la gespa i la sorra. Per crear tot això s'han usat diferents tècniques.

Primer creem un pla i li adjudiquem una textura d'asfalt, és important que sigui una textura repetitiva o *seamless texture*. Amb això ja tenim una superfície per la qual la *bubble* es podrà moure i ja podem començar a escriure l'*script* de moviment. Les textures que s'han usat en Bubble Racer són totes lliures i per a ús comercial.

Ara amb el Gimp creem el recorregut en planta; és recomanable usar l'eina camins. Un cop aplicada la textura a la superfície ja s'hauria de veure el recorregut, però aquest estarà pixelat. Podríem pensar que creant una imatge més gran no es pixelara, però això requeriria una mida enorme, i en conjunt, una mala solució. Per arreglar-ho creem un altre pla, amb un material de

un color blanc i el situem a la mateixa alçada que la superfície, però amb un marge suficient per no barrejar els materials en el *render*. Situem aquest segon pla sobre una recta en el recorregut i a base d'extrusions anem resseguint la forma del camí. Quan acabem podem unir els extrems del recorregut seleccionant els vèrtex corresponents i pressionant la tecla "F"*(Face)*. Per últim eliminem la textura sobrant de la superfície.

3.3. Creació del cel

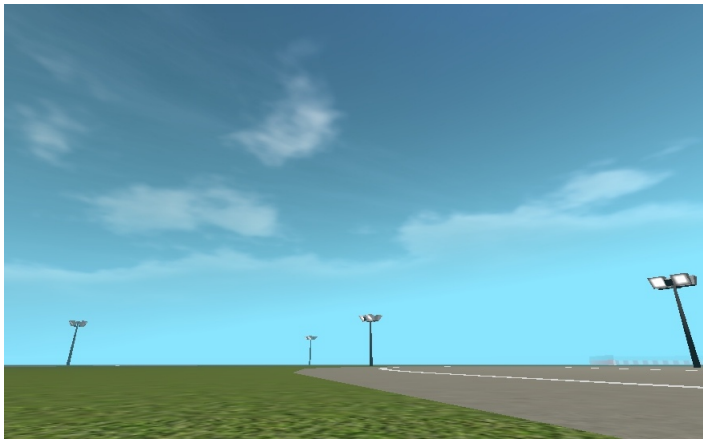
L'infinit és un problema. Si el joc ha de que processar tots els objectes de l'escenari, en el cas de que aquests siguin molts, es col·lapsara, fent alentir l'interpret i el sistema sencer. Per això molts *game engines* només *renderitzen* (dibuixen) fins a una certa distància de la càmera. En Blender aquesta distància la podem controlar a la càmera del nostra escenari, en el panell d'opcions. Dins el menú *object data*, a la pestanya *Lens*, canviant el valor de les caselles *start* i *end* de la opció *Clipping*. En Bubble Racer aquests valors són de 0.1 i 300 respectivament. El primer valor reflecteix la distancia mínima per començar a renderitzar, i el segon, la distancia màxima. Hem de tenir en compte que el cel i qualsevol altre element del paisatge no pot superar aquesta distància de la càmera, si volem que es vegi.

Un dels problemes en molts jocs antics és que els objectes allunyats, els que es surten del rang de renderitzat, apareixen "de cop" al entrar en aquest rang. Per solucionar-ho s'usen diferents tècniques. Una de les tècniques més modernes consisteix escanejar objectes reals amb punts de referencia, com si fossin àtoms. Aquests punts es poden reduir en la distancia per aixi crear gràfics de detall il·limitat[3]. També és possible crear objectes artificials a partir d'aquest objectes reals, i no es descarta en un futur la opció de importar malles poligonals. Aquesta tècnica està encara en desenvolupament, per la companyia Euclideon, però podria revolucionar en pocs anys el món dels videojocs. Com és lògic, aquest tipus de tecnologia punta no existeix en Blender ni en cap altre motor de joc actual, per això en el seu lloc s'acostuma a usar una tècnica més clàssica, difuminar els objectes llunyans, de la mateixa manera que fa qualsevol persona amb miopia. Malgrat que aquesta tècnica és molt agradable a la vista, és molt complicada de fer. En Blender disposem d'una opció per fer una tècnica similar de manera molt més fàcil. En aquest cas no difuminarem els objectes llunyans, sinó que els ocultarem en una boira blava, que es camuflara amb el cel.

3 Vídeo demostratiu de la tecnologia UD, absolutament increïble: http://www.euclideon.com/video01.html

Per fer això anem al menú *world* del panell d'opcions. Activem la pestanya *Mist* i immediatament apareixerà la boira esmentada. Podem configurar les opcions del rang de forma semblant a la càmera, en Bubble Racer aquestes opcions són *start: 10, Depth: 260*. A la pestanya *world* podem seleccionar el color de la boira i el color de fons. En el joc *horizon color* te un valor de *R: 0.251, G:0.773 B:0.985*. La seva equivalència en bits la trobem multiplicant per 255 de manera que RGB: 64, 197, 251. Podem obtenir les dades RGB de qualsevol editor de imatge normal.

Per ultim afegim una cúpula a l'escena amb unes proporcions de X:2.75, Y:2.75, Z:1. El fet de no ser perfectament esfèrica permetrà que els objectes desapareguin en la boira a l'horitzó, però que el cel sigui visible a partir de certa alçada. Perquè les dos cares d'un objecte siguin visibles ha de estar activada la opció *Backface culling* en el menú *materials* a la pestanya *game settings*. Afegim una textura panoràmica. Per ultim tenim que emparentar el cel amb la *bubble*, d'aquesta manera, allà on vagi la bombolla anirà el cel, mai el travessera.



Per crear un *parent* ho podem fer amb les tecles “ALT + P”, però sempre és més precís si ho fem des del menú *Object.* A la pestanya *Relations* seleccionem com a parent l'objecte que ens interessa, en aquest cas el pare és l'objecte “Dad”. La importància de tenir un objecte “Dad” recau a l'hora d'escriure l'script i ho explicaré més endavant. Hem de saber que existeixen tres tipus de parentesc*: vertex, 3 vertex*, i *object*. El primer només afecta a un vèrtex, és a dir, que l'objecte fill el seguira en posició, però no rotara com ell. Els dos següents són pràcticament iguals, només canvien en alguns espectres tècnics, però no en el seu funcionament. En els dos casos l'objecte fill segueix al pare, en posició en rotació i en grandària.

4. Python: BGE.

4.1. Panell de lògica

Com molts altres *Game Engines*, Blender compta amb algunes funcions a les que pots accedir sense programar. Aquestes funcions les trobem en el panell de lògica (*logic editor)*. Aquest panell es divideix en tres apartats: sensors, controladors, i actuadors. Els sensors serveixen per detectar. Es poden detectar col·lisions, variables, entrades del teclat, etc. Els controladors estableixen com i quan s'activara l'actuador si un sensor s'activa. L'actuador és el que fa l'acció, tal com fer moure objectes de l'escenari o reproduir determinats sons. En Python podem utilitzar aquests tres elements per crear programes senzills, o també podem fer les mateixes funcions utilitzant només línies de codi si volem crear programes més complexos. Ara bé, perquè s'activi el *game engine* en un script de Python necessitem afegir com a mínim un sensor i un actuador. Aquest són: un sensor del tipus *always* (sempre) i un controlador *Python* que executi l'script. Probablement existeixen més mètodes per arrencar un script en BGE, però per senzillesa i comoditat he usat aquest. Si volem crear un bucle a l'script ho tenim que fer activant la primera casella de l'opció de freqüència del sensor *always*. Els bucles són útils si volem que l'script sempre estigui actiu, però l'ús de bucles no deixa utilitzar variables locals en molts casos. En Python quan declarem una variable li tenim que assignar un valor, o aquest és assignat per defecte. És a dir, que en un bucle totes les variables que si declarin és sobreescriuran en cada passada. Per solucionar això o podem fer moltes maneres, per exemple, creant propietats d'objectes externs al bucle, o creant variables globals.

Les propietats d'objectes ens poden servir tant en els *scripts* com en els *logic bricks*. Per crear una propietat anem a la pestanya *propiertes* i clickem al botó a*dd game property*. Les propietats poden ser del mateix tipus que les variables: *string*, *float*, *intger* o *boolean;* o bé poden ser d'un tipus especial: *timer*. Un cop afegida la propietat en podem modificar el nom i els valors que venen donats per defecte. Pràcticament funciona igual que una variable global, amb la diferencia que les propietats pertanyen a un objecte, i només es poden cridar des d'aquest objecte en qüestió.

4.2. Experiments previs.

Els scripts i el seu funcionament han estat modificats i reescrits durant totes les fases de creació del videojoc. Això és a causa del descobriment constant de tècniques noves i la aplicació d'aquestes al codi. En un primer moment, vaig crear un script per cada objecte de l'escenari. Cada objecte tenia un sensor always i un script assignat a més de les propietats d'aquell objecte. Per intercanviar dades entre els scripts usava variables globals declarades en ells mateixos. A simple vista es veia que aquesta no era la forma correcte de programar, però a falta d'informació i d'una idea millor vaig prosseguir. Aquest primer intent va durar fins que vaig descobrir que els actuadors, els quals eren imprescindibles per al funcionament d'aquell script, podien enllaçar-se amb objectes diferents si seleccionàvem els objectes al mateix temps. Era una solució molt senzilla, semblava mentida que l'hages pogut passar per alt. Des de llavors només necessitava un script per governar-los a tots. Vaig crear un script per a cada apartat del programa, dividint-los en: arrencada, moviment, menú, interface i online.

Actualment he trobat un mètode encara millor, amb el qual ja no necessito de actuadors. L'ús d'actuadors simplificava molt la programació, però l'absència d'aquest permet una millor organització i independència a l'hora de programar.

4.3. Primer script

En molts llenguatges de programació el primer script és l'anomenat “Hola món” (*Hello world*). En Python 3 (Blender 2.6x) és tan fàcil com escriure la funció *print*, sense necessitat de importar llibreries. El resultat d'usar aquesta funció imprimira a la terminal el text especificat “Hello World”. Per comprovar-ho podem iniciar l'script prement el botó *Run Script* del panell de edició de text. Podem veure el resultat a la consola. En algunes versions de Blender la consola esta amagada, només tenim que obrir-la anant a *Help > Toggle System Console* del panell *Info*. En versions de Linux hem d'obrir Blender des d'una terminal (i no el rebés) per poder veure els resultats.

```
print("Hello World")
```

Nota: Blender treballa sobre un format que no permet l'ús d'accents gràfics, per tant, el codi font no pot tenir accents gràfics.

Aquesta funció és pròpia de Python i no requereix importar llibreries, però el motor de joc sí que necessita algunes llibreries. La llibreria de Blender, pel que fa referencia al game engine és la *bge* (*Blender Game Engine*). Per importar qualsevol llibreria usem la funció *import.* A diferencia de les llibreries pròpies de Python, les llibreries de Blender només estan disponibles quan iniciem el motor de joc, de manera que amb el botó "*Run Script*" ens saltaria un error conforme l'interpret no a trobat la llibreria. A continuació un exemple de script senzill de bge:

```
1 #Primer script, per Robert Planas.
2 import bge
3
4 controller = bge.logic.getCurrentController()
5 scene = bge.logic.getCurrentScene()
6 obj = controller.owner
7 obj.localPosition = [1,3,10]
```

A la primera línia del script trobem marcat en verd un comentari. Per crear un comentari escrivim #. Els comentaris són línies que no seran llegides per l'interpret i poden contenir qualsevol tipus d'informació, normalment orientada a entendre el codi. A la segona línia podem veure com s'importa la llibreria *bge*, tal i com s'ha explicat avanç. És comú en programació deixar algunes línies en blanc per separar estructures de codi i facilitar així la posterior feina d'anàlisis i correcció.

A la quarta línia creem una variable *controller* que conte una funció especifica. Això ens serà útil per no escriure cada vegada la funció sencera. Com podem veure les funcions estan separades per categories, podem accedir a cada categoria usant el punt. En aquest cas dins de la llibreria o biblioteca *bge* s'accedeix a la classe *logic* i a la funció *getCurrentController()*. Aquesta funció fa referencia al controlador actual. A continuació amb la variable *scene* podem veure el mateix, però en aquest cas especifica l'escena actual. A la sisena línia del codi veiem com usar la variable *controller* per abreviar funcions. En aquest cas *owner* fa referencia a l'objecte que executa l'script. És a dir, ara *obj* senyala a l'objecte que executa l'script. Ho podríem haver fet sense fer servir una variable *controller,* però la funció hages quedat més llarga: *obj = bge.logic.getCurrentController().owner*. De la mateixa manera podríem moure l'objecte com veiem a la línia 7.

No utilitzar abreviacions també és valid, però és recomanable l'ús de abreviatures per quan els scripts són grans, per no haver d'escriure sempre la funció sencera:

```
1 import bge
2 bge.logic.getCurrentController().owner.localPosition = [1,3,10]
3 bge.logic.getCurrentController().owner.orientation.z = 8
```

En aquest script podem veure l'exemple anterior sense usar abreviacions. A més s'ha afegit la funció *orientation*, en aquest cas especificant la coordenada z. Podem veure una llista completa de les funcions de la llibreria *bge* a la documentació de Blender, buscant l'estructura *bge.types*. Aquestes funcions corresponen a la classe encarregada dels objectes: *KX_GameObject*. A part de declarar un objecte amb la funció *owner*, que només ens permet declarar l'objecte des del que s'executa l'script, també ho podem fer des de la funció *objects* de l'escena actual, especificant el nom o la ID del objecte. La funció completa seria: *bge.logic.getCurrentScene().objects['Objecte']*.

```
1 import bge
2
3 cont = bge.logic.getCurrentController()
4
5 Act = cont.actuators["Actuador"]
6 Sen = cont.sensors["Sensor"]
7
8 cont.activate(Act) #Activa l'actuador.
```

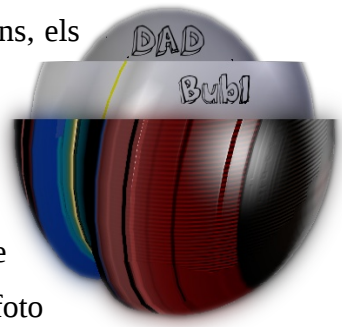
Podem combinar els scripts i el panell de lògica. Per fer-ho em de declarar els sensors i actuadors del nostre panell com en l'exemple anterior. Els actuadors estan normalment desactivats. Per activar-los podem fer servir la funció *activate* de la línia 8. Usant una estructura *If* podem activar i desactivar actuadors al nostre gust.

4.4. Moviment i rotació: Teoria

Per crear un moviment a la *bubble*, només cal que apliquem una força endavant o enrere en el moment que ens interessi. La direcció de la força ha de canviar quan la *bubble* giri, de manera que te que ser una força aplacada a una coordenada local, no global. Una coordenada local varia amb l'objecte (no ho fa amb la malla), quant aquest gira, totes les coordenades d'aquell objecte també. És a dir, que la coordenada Y sempre correspondrà a la part frontal de la *bubble*. Si les coordenades fossin globals aquestes es mantindrien sempre estàtiques sense importar el moviment o rotació de la *bubble*. Respecta la rotació a simple vista no hi ha cap problema, però aquest no és el cas a la realitat. Ens interessa que la *bubble* s'inclini, que giri i que acceleri. Totes aquestes rotacions actuen sobre coordenades locals, però cada acció mou aquestes coordenades de manera que, al intentar aplicar-ne més de una, la *bubble* es descontrola.

Per solucionar això s'ha creat un objecte anomenat *Dad* que s'encarrega de la posició i la direcció. Emparentat hi ha un objecte anomenat Bub1, el qual s'encarrega de d'inclinació. Per últim hi ha la *bubble*, que pot ser qualsevol de les 3 disponibles, i que s'encarrega de la rotació endavant o acceleració. Cada fill veu les seves coordenades locals influenciades per el pare, però el pare no és influenciat per les rotacions dels fills. D'aquesta manera podem crear les rotacions necessàries sense que l'objecte es descontroli.

A l'hora de implementar això al motor de joc tenim que tenir en compte altres factors. Primerament, l'objecte que rebrà les col·lisions. Pot ser qualsevol dels 3, però en aquest cas he escollit que sigui l'objecte Dad. Com que ja n'hi ha un que rep les col·lisions, els altres no ho tenen que fer, si ho fessin, al estar un dins del altre estarien col·lisionant constantment el que crearia inestabilitat. Un altre factor que hem de tenir en compte és que els objectes invisibles, el Dad i el Bub1 no han de tenir textura ni tampoc fa falta que tinguin malla, hi un tipus de primitiva sense malla anomenada *Empty* que ens pot ser molt útil. A la foto d'amunt podem veure una representació dels objectes esmentats.



4.5. Moviment i rotació: Script i lògica

Quan en el seu moment vaig realitzar l'script de moviment, no coneixia la funció *bge.logic.getCurrentScene().objects*, és per això que en el seu lloc vaig usar actuadors. Encara que el que explicaré a continuació es podria fer de la mateixa manera només amb codi, l'ús d'actuadors va crear alguns reptes difícils de solucionar. M'interessa mostrar com vaig solucionar aquests reptes per si algú es troba amb els mateixos problemes que jo. En aquest apartat, i en els que venen a continuació, només explicaré fragments del codi, el codi complert el podreu veure en els annexos. Els fragments estan dividits per les seves funcions, tals com la selecció de la *bubble*, el teclat, el moviment i la rotació.

4.5.1 Selecció de la *Bubble*

```
37  #Seleccio de Bubble.
38  Bub1 = cont.actuators['Bub1Visible']
39  Bub2 = cont.actuators['Bub2Visible']
40  Bub3 = cont.actuators['Bub3Visible']
41  if gl.globalDict["Bubble"] == 1:
42      cont.activate(Bub1)
43      Accel = 0.010
44      Fre = 0.015
45      Curv = 0.005
46      Reduct = 0.008
47      VelMax = 0.8
48      ActMov2 = cont.actuators['Motion2']
49      Servo.forceLimitY = [-30.0, 110.0, True]
```

Aquest és el codi que permet jugar amb una o un altra *bubble*. Com podem observar hi ha tres actuadors que corresponen a la visibilitat d'una o un altre *bubble*, que per defecte esta en invisible. A continuació trobem una sentencia condicional *if*. Aquesta sentencia dicta que: si la variable global *bubble* te un valor de 1 s'executara el codi següent, si no, no passara rés. En aquest tipus d'estructures és importantíssim respectar les tabulacions. Les condicions poden ser: igual que (==), més gran que (>), més petit que (<), igual o més gran que (=>) o igual o més petit que (=<). Les sentencies *if* també les podem acompanyar amb enllaços *and* i *or*. El primer executara el codi només si totes les condicions es compleixen, el segon l'executara quan qualsevol de les dos condicions es compleixi.

Dins la condició establim les coses que s'hauran de fer al seleccionar la *bubble* 1. En primer lloc aquesta es farà visible activant l'actuador corresponent. Després declarem una serie de variables que seran les propietats de la *bubble*, cada bombolla tindrà les seves pròpies propietats. Aquesta no és una programació orientada a objectes, de manera que les propietats es declaren com a variables globals. Aquestes són l'acceleració, l'acceleració de frenada, la pèrdua d'acceleració en les corbes, la quantitat de acceleració que es perdrà al superar una força especifica i la força màxima. Tots aquests valors només afecten a d'intensitat que s'aplicara a la força, no a la velocitat real. La forca màxima real es determina en la funció següent, a la línia 49. Aquesta funció fa referencia a un actuador del tipus *Motion* activat en mode *Servo Control*. En aquest cas especifiquem la força limit en l'eix Y. Especifiquem que és local, no des del script, sinó directament en el panell de lògica.

4.5.2 Teclat i acceleració

En el codi següent declarem les funcions corresponents al teclat:

```
10      keyboard = bge.logic.keyboard
11      JUST_ACTIVATED = bge.logic.KX_INPUT_ACTIVE
```

La segona funció rep el nom de JUST_ACTIVATED, encara que el seu nom no hauria de ser aquest. Aquestes constants especifiquen quan es considerara que s'ha pressionat la tecla. Les opcions disponibles són: KX_INPUT_NONE, KX_INPUT_JUST_ACTIVATED, KX_INPUT_ACTIVE i KX_INPUT_JUST_RELEASED.

```
80          if keyboard.events[bge.events.UPARROWKEY] == JUST_ACTIVATED:
81              own["acc"] = own["acc"] + Accel
82
83          if keyboard.events[bge.events.DOWNARROWKEY] == JUST_ACTIVATED:
84              own["acc"] = own["acc"] - Fre
85          if own["acc"] < 0:
86              own["acc"] = own["acc"] + 0.01
87              if own["acc"] > 0.001: #Control d'estabilitat de frenada.
88                  own["acc"] = 0
89          else:
90              own["acc"] = own["acc"] - 0.005
91              if own["acc"] < -0.001: #Control d'estabilitat de frenada.
92                  own["acc"] = 0
93          if own["acc"] > VelMax: # Velocitat maxima.
94              own["acc"] = VelMax
95          if own["acc"] < -0.3: # Velocitat maxima enrere.
96              own["acc"] = -0.3
97
98          if own["acc"] > 0.7: # Reductor de velocitat en limit.
99              own["acc"] = own["acc"] - Reduct #0.004
100         if own["acc"] > 0.8: # Reductor de velocitat en limit.
101             own["acc"] = own["acc"] - 0.001 #0.001
```

En aquest fregament podem veure com detectar la pulsació d'una tecla usant les variables que hem declarat anteriorment. Els primers dos condicionals són els que pròpiament detecten la tecla, mentre que els altres limiten la acceleració màxima o fan que la *bubble* s'estabilitzi quan la acceleració s'acosti molt al 0. En aquest exemple només podem veure el nom de les fletxes del teclat, però en la pàgina següent de la documentació de Blender podem trobar el noms de cada tecla:

http://www.blender.org/documentation/blender_python_api_2_59_0/bge.events.html

El nom *own* és l'abreviatura de *bge.logic.getCurrentController().owner*. Podem accedir a les propietats d'un objecte amb els símbols '[' i ']', i escrivint el nom de la propietat. A continuació podem veure l'impressor, és a dir, el codi que realment fa que totes aquestes variables s'apliquin o no.

4.5.3 Impressió de l'objecte “dad”

```
129         #IMPRESSIO
130         if SenSorra.positive == True: #Detecio de materials lents.
131             MatLent = True
132         if SenJespa.positive == True:
133             MatLent = True
134 
135         if MatLent == False:
136             if SenAsfalt.positive == False:
137                 Servo.linV = [0,own["acc"] /1.05 ,0]
138                 ActMov.dRot = [0,0,0]
139             if SenAsfalt.positive == True: #Detecio de vol.
140                 Servo.linV = [0,own["acc"],0]
141                 ActMov.dRot = [0.0,0.0,own["ori"]]
142 
143         if MatLent == True:
144             Servo.forceLimitY = [ -20.0, 20.0, True]
145             ActMov.dRot = [0.0,0.0,own["ori"]]
146             own["acc"] = own["acc"] / 1.04
147             Servo.linV = [0,own["acc"],0]
```

Tenint en compte que tot el codi de moviment és un bucle continu si, com és el cas, declarem una variable *MatLent = False* avanç de aquest codi, provocarem que la variable es reinicïi cada vegada, de manera que, excepte que una condició no la posi en *True*, sempre serà negativa.

Per acabar uns petits aclariments. L'actuador ActMov correspon al objecte dad, que en aquest codi només fa la funció de girar. Per la resta d'objectes s'usen les abreviatures ActMov2 que correspon a la *bubble*, i ActMov3 que correspon a l'objecte Bub1. Els sensors SenSorra, SenJespa i SenAsfalt no són del tipus *collision*[4] com es podria pensar, sinó que són de tipus *touch* amb un material assignat. Ressalto que, encara que assignis un material, el sensor actuara sobre tot l'objecte.

4.5.4 Rotació d'acceleració

```
168         #Rotacio endevant
169         acc2 = own["acc"]
170         acc2 = acc2/2
171 
172         if SenAsfalt.positive == False:
173             acc2 = acc2 * 4
174 
175         ActMov2.dRot = [-acc2,0.0,0.0]
```

Aquest fragment senzill no es distancia molt del que ja hem vist. Dividim l'acceleració perquè sembli que la *bubble* roti amb el terra. En cas que la *bubble* no toqui el terra, és a dir que voli, la acceleració no disminuirà, sinó que augmentara multiplicada per 2. Aprofito per dir que pots accedir a una dada de la matriu en concret, en lloc de especificar-les totes. D'aquesta manera el codi quedaria com: *ActMov2.dRot[0] = -acc2*. Hem de tenir en compte, però, que

4 Blender compte amb diferents tipus de sensors per a detectar altres objectes, un de col·lisions (*collision*), que funciona per propietats d'objecte, un te tocs (*touch*) per a materials, un de rang (*Near*) i un radar.

en alguns casos les matrius són coordenades. En aquest cas és correcte que per accedir a elles ho fem d'aquesta manera: *ActMov2.dRot[x] = -acc2*

4.5.5 Inclinació

Aquest a estat, sens dubte, el fragment que ha causat més problemes. Aquests problemes no esdevindrien no utilitzéssim actuadors, però en qualsevol cas vaig trobar una solució quant desconeixia aquest mètode. El problema es troba en que la rotació en l'actuador ha de ser local, per tant, mou els eixos. Si rotem l'objecte 0º aquest simplement no rotara, en canvi si el rotem 5º, aquest rotara aquests graus a cada passada del bucle. D'aquesta manera perdem el valor inicial, i quan necessitem que la *bubble* torni a la seva inclinació inicial, no sabem quants graus ha de girar. La solució ha estat guardar en una variable global anomenada *suma* els graus que es van sumant.

```
177         #Inclinacio
178         max = suma - gl.globalDict["Incl.Sumatori"]
179 
180         if control == 0: #Estabilitzador
181             if suma > 0:
182                 suma = suma - 1
183                 max = -1
184             if suma < 0:
185                 suma = suma + 1
186                 max = +1
187 
188         if suma >= 30: #Limits de velocitat
189             suma = suma - 1
190             max = 0
191         if suma <= -30:
192             suma = suma + 1
193             max = 0
194 
195         gl.globalDict["Incl.Sumatori"] = suma
196 
197         #IMPRESSIO (Inclinacio)
198         realpos = gl.globalDict["Incl.Max2"]
199         realpos = realpos + max
200 
201         gl.globalDict["Incl.Max"] = max / 100
202         gl.globalDict["Incl.Max2"] = realpos
203         ActMov3.dRot = [0,max/50,0]
```

Un altre possible solució hages estat crear un altre objecte el qual rotes uns graus concrets “de cop”, sense fer sumatori, i emparentar-lo amb un *slow parent* a la *bubble*. Aquest script que ara sembla tan simple em va costar esforços fer-lo ja que al principi no tenia idea de com fer l'estabilitzador. A més, en alguns casos els números no quedaven sincronitzats a causa de la ràpida pulsació de tecles. Això ho vaig arreglar amb la variable global *Incl.Sumatori* que correspon al valor de sumatori de la passada anterior.

4.6. Arrancada del joc.

El projecte esta dividit en tres escenes: *arranque*, *intro* i *game*. La primera escena s'encarrega de la resolució del joc, de crear les variables globals, i de mostrar un cop fet això, el vídeo d'introducció del joc. Un cop ha acabat el vídeo es passa a la segona escena, *intro*. Aquesta

s'encarrega del menú i de les configuracions. També s'encarrega de buscar connexions online: partides en xarxa i locals. Per ultim l'escena *game* és la pròpia de la pista i el joc.

4.6.1 Resolució configurable.

Blender no disposa d'una eina per canviar la resolució del joc en temps real[5]. Per el contrari el que tenim hem de fer és executar el joc des de la terminal del nostre sistema operatiu amb la ordre: *nomdeljoc.exe -f ResX ResY 16 60*, on Res correspon a la resolució, i X, i Y són les coordenades. Un exemple pràctic seria: *nomdeljoc.exe -f 1024 768 16 60*. Per automatitzar aquesta tasca podríem haver creat un programa tipus *"Launcher"*, que únicament es dediques a obrir el joc amb la resolució correcta. L'altre solució és obrir dos vegades el joc. El primer cop amb una resolució per defecte. Llavors el joc es reiniciaria automàticament amb els valors de resolució que toquen.

```
try:
    file = open('conf.txt')
    config = file.readlines()
    file.close()

    if config[2] == "Closed\n":
        file2 = open('conf.txt', 'w')
        file2.write(config[0] + config[1] + "Open\n" + config[3] + config[4])
        con0 = config[0].replace('ResX=','')
        con1 = config[1].replace('ResY=','')
        #file.close()
        os.system("start reset.bat " + con0.replace('\n','') + " " + con1.replace('\n',''))
        bge.logic.endGame()
    if config[2] == "Open\n":
        file3 = open('conf.txt', 'w')
        file3.write(config[0] + config[1] + "Closed\n" + config[3] + config[4])
        con0 = config[0].replace('ResX=','')
        con1 = config[1].replace('ResY=','')
        gl.globalDict["ResX"] = con0.replace('\n','')
        gl.globalDict["ResY"] = con1.replace('\n','')
        VS = config[3].replace('\n','')
        VM = config[4]
        gl.globalDict["VolumMusica"] = float(VM)
        gl.globalDict["VolumSo"] = float(VS)
        #file2.close()
except:
    print("No s'ha trobat l'arxiu conf.txt")
    gl.globalDict["VolumMusica"] = 1
    gl.globalDict["VolumSo"] = 0.8
```

En aquest script "*arranque*" usem la llibreria OS i les funcions pròpies de Python. Tot aquest apartat esta pensat per funcionar en Windows, per funcionar en altres sistemes operatius s'hauria d'adaptar[6].

5 A partir Blender 2.62 s'incorpora el canvi de resolució en temps real: *bge.render.setWindowSize()*

6 A partir del 20/10/2012 passo a treballar amb Linux (Ubuntu) i s'incorpora el canvi de resolució dinàmic.

A les línies 8, 9 i 10 obrim un arxiu i en guardem el contingut de cada línia a la matriu *config*[7], on cada línia és una coordenada de la matriu. Tot això ho fem dins de la estructura *try*, que en el cas de no trobar l'arxiu config.txt imprimira un error en pantalla i podrem seguir amb l'execució amb els valors per defecte, tal hi com veiem a partir de la línia 32.

A la tercera línia de l'arxiu *conf.txt* hi ha un valor que actua com a *boolean* i que pot ser *Open* o *Closed*. Malgrat que normalment aquests valors haurien de ser *True* o *False*, al estar actuant sobre un *string* podem assignar els noms que ens agradin més. Aquests dos valors indiquen si el joc ha canviat la seva resolució "*Open*", o si encara ho te que fer "*Closed*". Quant el valor és "*Closed*" es modifica l'arxiu canviant-lo a "*Open*". A continuació obté la resolució de l'arxiu de configuració i executa un programa en *batch* que s'encarrega d'obrir un altre cop el joc, però aquest cop amb la resolució que toca. Per no tenir dos copies del joc obertes, immediatament després d'executar el *batch* (línia 18), el joc es tenca.

Aquest és el contingut de *conf.txt* on els dos primers valors són la resolució, el tercer la variable *boolean* de la que ja hem parlat, i els dos últims els valors del volum.

```
ResX=1366
ResY=768
Closed
0.0
0.7
```

Aquest és el de *reset.bat*:

```
set x=%1
set y=%2
start gametest3.exe -f %x% %y% 16 60
Exit
```

4.6.2 Variables Globals.

Les variables globals les tenim que declarar en aquest script per dos motius. En primer lloc és l'script d'arrancada de manera que si les declarem aquí no ens sortira mai un error segons el

7 Una matriu és un conjunt de llistes. Les llistes (anomenades cadenes en casos d'agrupar lletres), són un conjunt de dades deferents, ordenades segons un index i guardades en una mateixa variable.

qual la variable a la que s'intenta accedir no ha estat declarada encara. En segon lloc aquest script només s'executa una vegada, de manera que això ens permet no haver de declarar-les en altres scripts. Als annexos podeu veure totes les variables globals declarades a l'script d'arrancada.

4.6.3 Textura de vídeo d'arrancada.

Blender Game Engine no suporta textures de vídeo per defecte. És a dir, que encara que en el mode de render podem afegir una textura de vídeo per reproduir un vídeo, en el BGE això no és possible fer-ho, sense Python. Per solucionar aquesta important falta, l'equip de Blender posa a disposició dels programadors de Python les textures dinàmiques, amb les quals, sí que és possible reproduir un vídeo en temps real. Podem trobar un exemple a la documentació de Blender:
http://www.blender.org/documentation/blender_python_api_2_62_1/bge.texture.html.

El següent script és el fragment corresponent al vídeo de Bubble Racer.

```
1  import bge
2
3  controller = bge.logic.getCurrentController()
4  scene = bge.logic.getCurrentScene()
5
6  obj = controller.owner
7
8  # Comprova que el video existeix.
9  if "Video" in obj:
10     video = obj["Video"]# Va a l'objecte video i el refresca.
11     video.refresh(True)
12
13 else:
14     #Crea l'objecte video (Dins de l'escena actual com si fos un material)
15     matID = bge.texture.materialID(obj, "MA" + "Pantalla")
16     video = bge.texture.Texture(obj, matID)
17     movie = bge.logic.expandPath('//title.avi')
18
19     video.source = bge.texture.VideoFFmpeg(movie)
20
21     video.source.scale = True # Escala el video
22     obj["Video"] = video #Crea l'objecte video.
23
24 video.source.play()
25
26 if obj["Temps"] > 2:
27     scene.replace("Intro")
```

Com podem veure tenim que estar refrescant el vídeo a cada passada del bucle, sinó aquest romandria com una imatge. La funció de la línia 15 ens permet accedir a la textura d'un material existent dintre l'objecte especificat. En aquest cas, el material “Pantalla” de l'objecte actual (*owner*). A continuació crea una textura dinàmica sobre aquest material. Especifiquem la ruta del vídeo que volem reproduir i després, a la línia 19, indiquem que serà una textura de vídeo. Finalment reproduïm el vídeo.

A la línia 26 veiem com, al cap d'un determinat temps, quan el vídeo finalitzi, canviarem de escena a l'escena del menú “Intro”.

4.7. Menú de selecció.

L'script de menús, amb més de 400 línies de codi, és sens dubte el més llarg de tot el joc. Malgrat que pugui semblar que és per la seva complexitat, en realitat és tan llarg a causa de que usa estructures repetitives necessitaries per crear el menú, i que no es poden simplificar fàcilment[8]. El funcionament d'aquests menús es basa en variables globals, concretament les que van de la línia 49 a la 56 del script de arrancada.

4.7.1 Menús per nivells.

Un grup de opcions o botons, pels quals ens podem moure és un nivell. La variable global que defineix en quin nivell ens trobem és *“MenuLabel”*. Com que és una variable *integer* això ens permetrà afegir tans nivells com desitgem. Per defecte li assignem el valor 1, corresponent al primer menú, que és el que inclou els botons de jugar, en xarxa i opcions. Quan polsem les tecles de pujar i baixar, actuem sobre 2 variables globals més. Aquestes són *“MenuPos”* i *“MenuEnd”*. La primera indica el botó en que ens trobem. En el nivell 1 el botó 1 correspon a “jugar”, el 2 a “en xarxa” i el 3 a “opcions”, ara bé, en cada nivell hi ha nous valors. Finalment la variable *“MenuEnd”* és la que indica la quantitat de botons que te el menú. En el cas del primer nivell aquest te 3 botons.

```
67      #Opcions de moviment amb les fletxes.
68      if keyboard.events[bge.events.DOWNARROWKEY] == JUST_ACTIVATED:
69          gl.globalDict["MenuPos"] = gl.globalDict["MenuPos"] + 1
70          if gl.globalDict["MenuPos"] == gl.globalDict["MenuEnd"] + 1:
71              gl.globalDict["MenuPos"] = 1
72
73      if keyboard.events[bge.events.UPARROWKEY] == JUST_ACTIVATED:
74          gl.globalDict["MenuPos"] = gl.globalDict["MenuPos"] - 1
75          if gl.globalDict["MenuPos"] == 0:
76              gl.globalDict["MenuPos"] = gl.globalDict["MenuEnd"]
77
78      if keyboard.events[bge.events.RIGHTARROWKEY] == JUST_ACTIVATED:
79          gl.globalDict["SelectPos"] = gl.globalDict["SelectPos"] + 1
80          if gl.globalDict["SelectPos"] == gl.globalDict["SelectEnd"] + 1:
81              gl.globalDict["SelectPos"] = 1
82
83      if keyboard.events[bge.events.LEFTARROWKEY] == JUST_ACTIVATED:
84          gl.globalDict["SelectPos"] = gl.globalDict["SelectPos"] - 1
85          if gl.globalDict["SelectPos"] == 0:
86              gl.globalDict["SelectPos"] = gl.globalDict["SelectEnd"]
```

Quan ens trobem sobre un menú de selecció, en el qual podem escollir diverses opcions usem les tecles dreta i esquerra. Aquestes opcions de selecció funcionen de forma similar a les de menú, però requereixen de variables noves. Aquestes variables són *“SelectPos”* i

8

Una manera de simplificar-les seria usant POO. En el futur tinc pensat crear una llibreria per simplificar tasques com aquesta.

"SelectEnd". Optativa-ment podem usar també les variables *"PreMenuPos"* i *"PreSelectPos"* les quals ens seran útils a l'hora de crear animacions, per saber els valors anteriors de *"MenuPos"* i *"SelectPos"* respectivament.

El fragment anterior, per a molts programadors hauria de estar dins una funció o una rutina. De fet estic segur que aquesta no és la millor forma de programar, una forma no orientada a objectes (POO) i sense utilitzar funcions. Però ja que aquest script es te que executar igualment com un bucle, he decidit no crear-li una funció o rutina, en el seu lloc he usat variables globals booleans per activar o desactivar un codi mitjanant una estructura *if*. Un exemple d'això el podem trobar a la línia 35 de l'script "*interface*".

4.7.2 Animacions i funcions dels menús.

Mitjançant més estructures *if* podem crear animacions als menús, tal i com podem veure en el codi següent. El que s'intenta és que per un valor de *MenuPos* s'assignin els moviments corresponents a cada boto. Hem de tenir en compte que en alguns casos els botons no canvien de posició. En aquests moments ens saltem la funció per simplificar el codi.

```
89      if gl.globalDict["MenuLabel"] == 1: #Menu d'inici.
90
91          #Animacions dels botons.
92          if keyboard.events[bge.events.DOWNARROWKEY] == JUST_ACTIVATED or \
93          keyboard.events[bge.events.UPARROWKEY] == JUST_ACTIVATED:
94              if gl.globalDict["MenuPos"] == 1:
95                  JugarOBJ.localPosition.x = -0.9
96                  XarxaOBJ.localPosition.x = -1.1
97                  OpcionsOBJ.localPosition.x = -1.1
98              if gl.globalDict["MenuPos"] == 2:
99                  JugarOBJ.localPosition.x = -1.1
100                 XarxaOBJ.localPosition.x = -0.9
101                 OpcionsOBJ.localPosition.x = -1.1
102             if gl.globalDict["MenuPos"] == 3:
103                 JugarOBJ.localPosition.x = -1.1
104                 XarxaOBJ.localPosition.x = -1.1
105                 OpcionsOBJ.localPosition.x = -0.9
106             gl.globalDict["PreMenuPos"] = gl.globalDict["MenuPos"]
```

A continuació podem veure les funcions de cada botó. Malgrat que en diem funcions amb el benentès que serveixen per fer funcionar alguna cosa, no son realment funcions en programació, ja que no han estat declarades com a tal. Per alguns programadors, el correcte hauria estat crear una funció per cada boto, aixi en el cas de tenir dos botons que hagessin de fer el mateix, l'únic que haurien de fer seria executar la mateixa funció. En aquest menú, però, hi ha un únic boto per cada funció, de manera que resulta inútil crear-les.

Recordo que per crear una funció usem l'estructura *def nom(arguments)*. Podem cridar una funció amb l'estructura *nom(paràmetres)*. Les propietats són fonamentals en les funcions. Podem veuen un exemple en la funció per processar el temps de l'script "*interface*". En aquest cas, sí que ha estat útil per simplificar el codi. Recomano veure el fragment sencer des de la línia 88 a la 140. (Bubble Racer v0.2)

```
131     if gl.globalDict["MejorTiempo2"] == 99999:
132         printText4.value = '"00:00.00"'
133
134 def ProcesarTiempo(tiempo):
135     minutos = int(tiempo/60)
136     segundos = int(tiempo - minutos*60)
137     decimes = int((tiempo - segundos - minutos*60) * 100)
138     tiempo = str(minutos) + ":" + str("%02i"% segundos) + "." + str(decimes)
139     return tiempo
140 main()
```

Com podem veure la funció *ProcesarTiempo* s'encarrega de processar la variable *tiempo*. A l'ultima línia cridem la funció *main*, dons tot l'script "*interface*" s'executa dins d'aquesta funció. La funció *main* necessita ser cridada perquè s'executi, ja que *main* en Python només és un nom, no te cap propietat especial que impliqui que aquesta és la funció principal a executar.

```
108         #Accions dels botons.
109         if keyboard.events[bge.events.RETKEY] == JUST_ACTIVATED:
110             if gl.globalDict["MenuPos"] == 1:
111                 gl.globalDict["MenuLabel"] = 3
112                 gl.globalDict["PreMenuPos"] = 1
113                 gl.globalDict["MenuPos"] = 1
114                 EmptyOBJ.localPosition.y = 0
115                 gl.globalDict["SelectLabel"] = 1
116                 gl.globalDict["MenuEnd"] = 2
```

L'script superior correspon a l'acció de canviar de menú, de *MenuLabel* quan pressionem la tecla "*Enter*" estant en *MenuPos = 3*. No només tenim que canviar el valor de *MenuLabel*, sinó que també hem d'establir les propietats corresponents al menú que volem accedir.

Per acabar, aquí podem veure els fragments corresponents a la selecció de la resolució i al botó desar, encarregat de guardar els canvis a l'arxiu *conf.txt* i de reiniciar el joc.

```
261         if gl.globalDict["MenuPos"] == 4:
262
263             if keyboard.events[bge.events.RETKEY] == JUST_ACTIVATED:
264
265                 file2 = open('conf.txt', 'w')
266
267                 file2text = 'ResX=' + gl.globalDict["ResX"] + '\n' + "ResY=" + gl.globalDict["ResY"] + \
268                 '\n' + 'Closed\n' + str(gl.globalDict["VolumMusica"]) + '\n' + str(gl.globalDict["VolumSo"])
269                 file2.write(file2text)
270                 file2.close()
271                 os.system("start reset.bat " + gl.globalDict["ResX"] + " " + gl.globalDict["ResY"])
272                 bge.logic.endGame()
```

```
        if gl.globalDict["MenuPos"] == 1: #Resolucio

            if keyboard.events[bge.events.LEFTARROWKEY] == JUST_ACTIVATED or \
            keyboard.events[bge.events.RIGHTARROWKEY] == JUST_ACTIVATED:

                if gl.globalDict["SelectPos"] == 1:
                    ResX = 720
                    ResY = 480
                if gl.globalDict["SelectPos"] == 2:
                    ResX = 800
                    ResY = 600
                if gl.globalDict["SelectPos"] == 3:
                    ResX = 1024
                    ResY = 720
                if gl.globalDict["SelectPos"] == 4:
                    ResX = 1024
                    ResY = 768
                if gl.globalDict["SelectPos"] == 5:
                    ResX = 1280
                    ResY = 720
                if gl.globalDict["SelectPos"] == 6:
                    ResX = 1280
                    ResY = 768
                if gl.globalDict["SelectPos"] == 7:
                    ResX = 1360
                    ResY = 768
                if gl.globalDict["SelectPos"] == 8:
                    ResX = 1366
                    ResY = 768

                resolution.value = '"' + str(ResX) + 'x' + str(ResY) + '"'
                gl.globalDict["ResX"] = str(ResX)
                gl.globalDict["ResY"] = str(ResY)
                gl.globalDict["PreSelectPos"] = gl.globalDict["SelectPos"]
                gl.globalDict["SelectEnd"] = 8

```

El signe \ al final d'una línia serveix per separar en línies ordres massa llargues.

5. Python: Mode Online

En el mode *online* (en línia), o com expresso en Bubble Racer “En xarxa”, existeixen dos modes de jugar a traves de la xarxa. El primer, i el més conegut, a traves d'Internet, el segon, a treves d'una xarxa local. Tots a casa tenim xarxes locals, però aquest nom és poc conegut. Ara bé, el nom “*wifi*”, més conegut, ens pot servir per entendre el que és una xarxa local. Qualsevol aparell connectat a un altre mitjançant un tercer, el qual administra les connexions, forma una xarxa. Hi ha molts tipus de xarxes, però totes coincideixen en que cada aparell connectat a la xarxa ha de tenir un numero o codi identificatiu, aquest en el món d'Internet s'anomena IP. Internet no és més que un altre xarxa, però en aquest cas mundial.

5.1 Teoria de les xarxes i Internet.

Entenent lo anterior ens adonem que, si tenim un ordenador connectat de forma indirecta, mitjanant *wifi* a Internet, aquest en realitat esta connectat a dos xarxes. Com que hi ha dos xarxes, també hi ha dos IP. La IP local i la IP pública. La IP local és la pròpia de l'ordenador, o més ven dit, del adaptador de xarxa.

5.1.1 Tipus d'IPs.

Hi ha moltes maneres de conèixer una IP local, la més comuna en un Windows és executar el programa “cmd.exe”, per exemple, des de la finestra executar del menú Inici, i escriure el comandament *ipconfig*. Apareixeran totes les IP del teu ordenador, una per adaptador de xarxa, ja sigui *wifi*, per cable o artificial. En canvi, per conèixer una IP pública no ho podem fer des de la xarxa local, per aconseguir-ho hem de usar servidors externs que ofereixin aquest servei. El servidor de Bubble Racer compta amb aquest servei perquè els usuaris puguin disposar de les seves IP immediatament i sense complicacions.

Aquest tipus de connexions, de caràcter domèstic, estan connectades entre elles mitjanant un enrutador[9] o *router*. És a dir, que per poder tenir accés a la xarxa local o a Internet, abans t'has de connectar a la IP aquest aparell, amb IP pròpia. Aquest tipus d'IP s'anomena “Porta d'enllaç”. Per defecte aquesta sol ser 192.168.1.1, mentre que les IP de cada aparell solen estar entre 192.168.1.10 i 192.168.1.255.

Cada numero separat per un punt correspon a un byte. Els bytes són agrupacions de 8 bits i cada bit representa un valor de apagat (0) o encès (1). D'aquesta manera la combinació de 8 bits dona com a resultat un numero màxim de possibilitats: 2^8= 256. És per això que els valors de les Ipv4 són sempre d'entre 0 i 255.

Encara que 256 possibilitats són més que suficients per una xarxa domèstica, en grans empresses pot resultar insuficient. A més, tenint en compte que cada connexió a Internet requereix una IP, el numero de ordenadors que es poden connectar a Internet mitjanant aquest tipus de IP no és infinit. Com avanç, podem calcular que el numero màxim de IPs que poden tenir accés a Internet simultàniament és de 256^4 = 4.294.967.296. És a dir, 4.300 milions d'aparells aproximadament, tenint en compte que al món som unes 7.000 milions de persones i que, malgrat que no tothom te Internet, molta gent te més d'una connexió (casa, mòbil, feina) i que les grans empreses usen moltes IPs, és normal pensar que no falta molt perquè arribem al límit. És per aquest motiu que, últimament i cada cop més, s'esta aplicant el nou protocol de IP, l'Ipv6.

9 No confondre router amb mòdem. Un mòdem només permet una connexió directa a Internet, només te una IP.

IPv6 estructura els bits d'una forma completament diferent. En aquest cas ho fa en grups de 4 bits, és a dir: 2^4 = 16, que funcionen com a un sistema numèric nou, diferent al sistema numèric comú decimal en base 10 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). En aquest cas es un sistema numèric hexadecimal en base 16 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f). Podem trobar-ne més informació a l a web: http://ca.wikipedia.org/wiki/Sistema_hexadecimal. En el cas de les IPv6 les separacions, anomenades grups, no són per punts, sinó per dos punts (:) i cada grup pot contenir un valor de 4 dígits hexadecimals, és a dir: 16^4 = 65536 possibilitats. Amb un total de 8 grups, el total d'IPs possibles és de 16^4^8, més de 340 sextrillons de possibilitats. Ara bé, els 4 últims grups normalment corresponen, no a l'adreça del aparell connectat a Internet, sinó a la direcció MAC de l'aparell de dins la xarxa local que esta sol·licitant la connexió. En altres paraules, encara que en IPv4 tots els ordenadors d'una xarxa local tenien la mateixa IP pública, en Ipv6 cada aparell, independentment de la xarxa en la que es trobi, posseeix una IP pròpia.

Un altre factor a tenir en compte, sobretot a l'hora de configurar una xarxa o crear un servidor a Internet, és l'estabilitat de la IP. Cada cop que ens connectem al *router*, o que aquest es connecta a Internet, escollim una IP. A l'hora d'escollir IP tenim dos opcions.

La primera opció, i la més habitual, és que l'ordenador demani al *router* que li assigni qualsevol IP que estigui disponible. Aquest es fa mitjançant DHCP. La nostra IP pública esta configurada per defecte d'aquesta manera. Quan un adaptador es comporta així diem que obté una IP dinàmica. L'altra opció és la IP fixa, és a dir, l'ordenador s'intenta connectar sempre amb la mateixa IP. Si aquesta esta ocupada per algú altre pot donar un error de conflicte de IPs, però es pot configurar el sistema perquè, en cas de estar la IP ocupada, intenti connectar-se amb un altre IP, o utilitzi una IP dinàmica.

5.1.2 Ports

Un port és un canal d'informació entre dos aparells informàtics. És a dir, “l'entrada dels cables”. D'aquí ve quan diem “el port USB”. A diferència dels ports físics, un *roter* te dos tipus de ports amb connexió a la xarxa. Els ports físics “LAN” i els ports lògics. Els ports lògics són els que s'encarreguen de transmetre la informació per la xarxa local i serveixen bàsicament per identificar l'aplicació a la qual s'esta enviant d'informació. Per controlar aquest

tipus de port ho podem fer usant el nostre navegador preferit i accedint a la porta d'enllaç a la barra d'adreces. És a dir, en lloc d'escriure *"www.google.com"* escrivim "192.168.1.1". Cada *roter* te un *software* propi del seu model, però tots han de permetre obrir i tancar ports lògics.

Perquè una connexió pugui establir-se entre dos punts de la xarxa local s'ha de especificar el port, i , a més, aquest ha d'estar obert. Existeixen alguns ports predefinits, per exemple, el port que s'encarrega de la navegació per pàgines web, el 80. Els ports poden ser del 1 al 65536. Alguns ports venen oberts per defecte, podem saber quins són des de la terminal de windows teclejant *netstat -an | find "LISTEN"* i des de Linux amb *sudo netstat -lp --inet*. Si hi ha ports oberts des del *router* per a totes les IP, aquest s'especifiquen com 0.0.0.0 o [::] en el cas de IPv6.

5.2 Sockets

Els *sockets* ens permeten, especificant un port i una IP, establir una connexió amb un ordenador i enviar-li dades. En Python podem utilitzar *sockets* important primer la llibreria *socket*. Dins la llibreria *socket* podem trobar diverses ordres per crear connexions i enviar dades. A Internet la informació sobre els *sockets* en Python és abundant, però gran part d'aquesta es troba en Python 2, Blender 2.61 usa Python 3.

5.2.1 L'"Hola Món" del *socket*.

Per establir una connexió necessitem com a mínim un client i un servidor. El client s'encarregara de connectar-se al servidor i enviar-li informació. El servidor s'encarregarà de establir la connexió i d'imprimir en pantalla el text enviat pel client. A continuació un exemple de client.

```
1  import socket
2
3  sock = socket.socket()
4  sock.connect(("localhost", 6080))
5
6  while 1:
7      missatge = input("-> ")
8      missatgeUTF = bytes(missatge, 'UTF-8')
9      sock.send(missatgeUTF)
10     print("Missatge enviat:" + missatge)
11     if missatge.count("quit") == 1:
12         print("Sortint del bucle...")
13         break
14
15 sock.close()
16 print("Fi del programa.")
```

El primer que fem és crear l'objecte *sock* amb la funció *socket.socket()*. A continuació ens connectem al servidor amb la IP *"localhost"*. Quan en lloc d'una IP escrivim una adreça, aquesta serà convertida en IP si és possible, mitjançant DNS. Això ho podem comprovar si fem *ping www.google.com* la IP resultant correspondrà al servidor de Google en el que estiguem connectats dins dels més de 450.000 que les últimes estimacions diuen que posseeix. La direcció *localhost* és una abreviatura que correspon a la IP 127.0.0.1, que s'usa per a operacions internes del ordenador, a continuació especifiquem el port. Els ports tenen que estar oberts tan en el router com en el talla focs, exceptuant que fem connexions amb el nostre propi ordenador usant *localhost*.

A continuació obrim un bucle *while* infinit. La primera variable *missatge* guardara el valor de la funció *input*, el valor que entrem a la consola del sistema. Fins que no teclegem un valor a la consola l'script quedara interromput. A la següent línia convertim aquest valor *string* en binari especificant una codificació UTF-8, que no permet accents ni caràcters estranys, el resultat d'aquesta conversió el guardarem en *missatgeUTF*. Tot seguit enviem el missatge codificat. En Python 3 és obligatori enviar les dades en binari. Si el missatge és "*quit*" trencarem el bucle i finalitzarem el programa. La funció *count* retorna el valor *integer* corresponent a la quantitat de vegades que es repeteix una cadena dins d'un altre, en aquest cas *missatge*. Per buscar la posició podem usar *find*.

A continuació podem veure un exemple de servidor, el qual s'encerregara de crear la connexió, escoltar els missatges a rebre, i imprimir-los en pantalla.

```
import socket

sock = socket.socket()
sock.bind(("localhost", 6080))
sock.listen(1)

print("Esperant connexio...")
socket_client, dades_client = sock.accept()
print("Conectat amb: " + dades_client[0] + ":" + str(dades_client[1]))

while 1:
    print("Esperant missatge...")
    dades_rebudes = socket_client.recv(1000)
    missatge = str(dades_rebudes)
    print("El missatgee es:" + missatge)

    if missatge.count("quit") == 1:
        break

print("Tencant...")
socket_client.close()
sock.close()
```

Aquest codi és molt similar el exemple anterior però en aquest cas usem la funció *sock.listen(1)* per escoltar connexions entrants, acceptant com a molt 1 connexió. Un cop acceptada la connexió en guardem les dades a les variables *socket_client* i *dades_client*. El primer correspon al socket del client i en posseïx les funcions com si d'un socket propi es tractes. La segona és una matriu amb la IP i port del client.

Dins el bucle les coses tornen a canviar. En lloc de enviar un missatge el rebrem, per això usem la funció *recv()* del socket client. Especificant que rebrem fins a un màxim de 1000 bytes. Les dades rebudes en bytes les convertim en text amb la funció *str()* i en guardem el resultat en una variable *string* anomenada “missatge”.

Els missatges que s'imprimeixen en la terminal de Windows del programa servidor poden contenir alguns caràcters estranys a principi i final de línia. Això és degut al canvi de codificació del UTF-8 de la variable al ASCII de la consola. Aquest error no succeïx en Linux, dons la consola de Linux és UTF-8.

5.2.2 Xat amb múltiples clients.

Ara intentarem crear un xat a traves de sockets. Per crear el xat crearem dos programes, un client i un servidor, però en aquest cas funcionaran de forma completament diferent al exemple anterior. El servidor s'encerregarà d'escoltar i acceptar connexions, els clients enviaran missatges al servidor i aquest al mateix temps retornara el missatge a la resta de clients. A continuació podem veure el codi complet que correspon al servidor. Com a novetat he usat la llibreria *select* a més de la llibreria *socket* que ja hem vist anteriorment. La llibreria *select* resulta útil a l'hora de crear llistes o taules que utilitzarem per guardar la informació de cada *socket* que es connecti. Però més endavant veurem que pot resultar prescindible.

```
4  def accept_new_connection():
5
6      try:
7          global server
8          global desc
9          newsock, (remhost, remport) = server.accept()
10         server.settimeout(.1)
11         print("Se ha conectado %s:%s" % (str(remhost), str(remport)))
12         desc.append(newsock)
13     except:
14         pass
15
16 def broadcast(msg, sock):
17
18     global desc
19     global server
20     host, port = sock.getpeername()
21     msg = "[%s:%s]: %s" % (str(host), str(port), str(msg))
22     for destsock in desc:
23         if destsock != sock and destsock != server:
24             missatgeUTF = bytes(msg, 'UTF-8')
25             destsock.send(missatgeUTF)
26
27 def get_msg(sock):
28
29     try:
30         msg = sock.recv(1024)
31         sock.settimeout(.1)
32         return msg
33     except:
34         global desc
35         host, port = sock.getpeername()
36         print("[%s:%s] ha salido." % (str(host), str(port)))
37         desc.remove(sock)
38         return None
39
40 global server
41 server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
42 server.bind(("", 6080)) #IP & Port
43 server.listen(5)
44 print("Iniciado servidor. Esperando connexiones en el puerto 8000...")
45 global desc
46 desc = [server]
47 while 1:
48     accept_new_connection()
49     (sread, swrite, sexc) = select.select(desc, [], [])
50     for sock in sread:
51         if sock != server:
52             flag = get_msg(sock)
53             if flag:
54                 broadcast(flag, sock)
```

El codi inicia definint tres funcions, és a dir que realment el codi comença a partir de la línia 40.

Com anteriorment, creem el *socket*, però en aquest cas ens posem a escoltar qualsevol IP des de el port 6080 i acceptem fins a un màxim de 5 connexions per aquest *socket*. A continuació imprimim un missatge conforme el programa a iniciat correctament. Dins d'un bucle executem la funció per acceptar noves connexions. Si algú es connecta en menys de 0.1 segons es guardaran les dades a les variables corresponents i s'imprimira el missatge en pantalla. En cas contrari es creara una excepció que simplement acabara amb la funció i seguirem a la següent instrucció del bucle. És aquí on utilitzem la llibreria *select* per guardar en una taula les dades dels diferents *sockets* connectats. Tot seguit, mitjanant una estructura *for,* executem les instruccions següents tantes vegades com *sockets* hi hagi a la taula. Iniciem llavors la funció *get_msg,* que escoltara durant un curt període de temps si hi ha algun missatge des de aquella IP. Si no és el cas, pot ser que simplement no s'hagi enviat rés, aleshores acabem la funció i reiniciem el bucle o el *for*.

També és possible que es produeixi una excepció. En aquest cas s'elimina el socket i el registre d'aquest, i s'informa a l'usuari que un client s'ha desconnectat. Si el missatge s'ha rebut correctament aquest es guarda a la variable *flag*. Després d'un *if* que comprova si la variable esta buida, executem la funció *boardcast* encarregada de codificar i reenviar el missatge a la resta de clients.

```
import sys
import socket
import select
import threading

class obj_tel(object):

    print("---------CLIENT---------")

    def __init__(self):

        self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            self.client.connect(("localhost", 6080))
            print("\n Conetado con el servidor. \n")
        except:
            print("\nNo se ha podido connectar.\n")
            sys.stdin.readline()
            self.exit_client()
        self.telnet()

    def telnet(self):

        self.opt = [self.client]
        self.msg = ''
        while 1:
            (self.rlist, self.wlist, self.xlist) = select.select(self.opt, [], [], .9)
            if self.rlist != []:
                print("%s" % (str(self.client.recv(1024)), ))
            else:
                self.slp = threading.Thread(target = self.get_data)
                self.slp.start()

    def get_data(self):
        self.tmp = str(sys.stdin.readline())
        self.msg = ''
        for self.i in self.tmp:
            if self.i != '\n':
                self.msg += self.i
        missatgeUTF = bytes(self.msg, 'UTF-8')
        self.client.send(missatgeUTF)

    def exit_client(self):
        raise SystemExit

if __name__ == "__main__":
    telnet = obj_tel()
```

En aquest cas també tenim noves llibreries, aquestes són *threading* i *sys*. La llibreria *sys* serveix per enviar ordres com si ho fessis des de la consola. La llibreria *threading* serveix per crear fils d'execució, això ens permetrà, a efectes pràctics, executar més d'una instrucció al mateix temps.

Iniciem la classe *obj_tel* des de la funció *__init__*, aquest nom en clau executara la funció com a funció d'arrencada. És important recordar que les funcions de les llibreries importades no es poden usar en classes si no estan dins d'una funció. Com sempre el primer que fem és crear el *socket* i connectant-se al servidor.

A continuació entrem a la funció *telnet* encarregada d'escoltar i imprimir per pantalla les dades que rebi del servidor. Tanmateix aquesta funció, en el cas de no rebre dades, inicia un nou fil, aquest fil esta emmarcat en la funció *get_data*, encarregada de codificar i enviar missatges al servidor.

L'exemple anterior pot ser complicat d'entendre si no es coneix el funcionament de cada funció o variable del programa, per això a continuació deixo una taula amb una petita definició de les més importants.

Sys.stdlin.readline()	S'encarrega de llegir línies de la consola del SO. Permet introduir dades que, en aquest cas, són guardades a la variable *self.tmp*
Raise SystemExit	Tanca el programa. Això també tancarà Blender, és dons similar a *bge.logic.endGame()* però en aquest cas no requereix de la llibreria BGE, només disponible en l'execució del motor de joc.
Trheading.Trhead()	Serveix per declarar un fil d'execució. Te diferents funcionaments, en aquest cas s'ha usat *target*.
TrheadObject.start()	Inicia l'execució del fil.
Self.Opt	Guarda la informació del socket; IP i Port. Self fa referencia a l'objecte *self*, declarat a començament de la classe.
'\n'	S'usa per especificar un salt de línia.
"%s"	Prové de Python 2 però també és molt usat en altres llenguatges. S'usa per substituir el valor d'una cadena que enganxem a continuació amb el signe %.
Socket.SOCK_STREAM	Ens permet definir el tipus de socket, en aquest cas usem protocol TCP.
Socket.AF_INET	Ens permet definir el tipus de socket, en aquest cas usem protocol IPv4.

Com hem vist a l'exemple anterior comptem amb forces utilitats que, pel que nostres volem, ens resulten inútils. És per això que ara tenim que simplificar el codi i adaptar-lo a les nostres necessitats. Per exemple eliminarem el codi de la línia 37 a la 39, ja que no necessitem fer salts de línia en els nostres missatges. També simplificarem el codi eliminant la classe, o eliminarem notificacions innecessaris. Després de reduir el codi, tocara ampliar-lo una altra vegada, però en aquest cas implementant la BGE de manera que ja pugem veure els resultats reals del mode en línia.

5.2.3 Implementant la BGE.

Un dels problemes més importants del treball de recerca és la implementació del xat anterior juntament amb el motor de joc de Blender. Fins al moment em executat l'script només des de

la consola de Python, sense tenir en compte la BGE, però si intentem executar l'scrpt sense modificar, juntament amb la BGE, aquesta es col·lapsa. Això succeïx perquè, tot i els *threads*, la funció principal esta reiteradament en un bucle, sense deixar que finalitzi l'script i s'executi el motor de render de BGE. Podem arreglar-ho modificant l'script perquè els bucles es creïn de forma independent, fora d'una classe i en els seus fils corresponents. D'aquesta manera al finalitzar l'script s'iniciara el renderitzat.

```
import threading

def func1():
    while 1:
        print("Thread 1")
def func2():
    while 2:
        print("Thread 2")

Thr1 = threading.Thread(target = func1)
Thr1.start()
Thr2 = threading.Thread(target = func2)
Thr2.start()
```

Ara bé, hi ha un detall que no hem tingut en compte. El motor de joc no deixara executar els fils si l'script en el que es troben no esta actiu, és a dir, estem obligats a crear un bucle a l'script des de els *logic bricks*. Això, però, creara de forma recurrent, un munt de fils nous dins de bucles infinits que de segur acabaran amb la memòria RAM del ordenador. Per evitar-ho tenim que crear una variable global o una propietat booleana que, mitjançant un *if*, ens serveixi per executar una sola vegada la part d'script que ens interessa. A continuació podem veure el codi anterior apte per funcionar juntament amb BGE. Ara ja sí que podem implementar el xat al BGE, l'enorme script el podeu trobar als annexos.

```
import bge
import threading

def func1():
    while 1:
        print("Thread 1")
def func2():
    while 2:
        print("Thread 2")

own = bge.logic.getCurrentController().owner

if own["Ones"] == False:
    Thr1 = threading.Thread(target = func1)
    Thr1.start()
    Thr2 = threading.Thread(target = func2)
    Thr2.start()
    own["Ones"] = True
```

5.2.4 Implementant la BGE II.

Per a simplificar el codi he creat un arxiu anomenat “internet.py” que conte les funcions encerregades de connectar-se amb el servidor i enviar dades. Aquest arxiu funciona com una

llibreria externa, per usar-lo escrivim *import internet*. Una de les condicions que ha de tenir aquest fitxer perquè se'l pugui importar és la extensió *.py*. Blender te un *bug* amb aquest tipus d'extensió de manera que és millor editar l'arxiu sempre externament amb un editor com *notepad++* o *gedit*.

Ara toca fer que les dades guardades a la funció *var* siguin traslladades al videojoc. Per a això s'ha utilitzat el codi següent:

```
def OnlineMode():

    CargarOnline()

    #RENDER OthrPOST
    for a in range(0,internet.var.ID_Num):

        a = a+1

        if not a == internet.var.ID:
            try:
                gl.OnlineOBJ[a].worldPosition = [int(internet.var.PosX[a])/100, int(internet.var.Pos\
                OnlOri = gl.OnlineOBJ[a].localOrientation.to_euler()


                OnlOri.x = int(internet.var.RotX[a])/100
                OnlOri.y = int(internet.var.RotY[a])/100
                OnlOri.z = int(internet.var.RotZ[a])/100
                gl.OnlineOBJ[a].localOrientation = OnlOri
            except:
                print("FError?")
    #Send MyPOST
    BubOri = Bub1.worldOrientation.to_euler()

    internet.var.PosX[internet.var.ID] = int(Dad.worldPosition.x*100)
    internet.var.PosY[internet.var.ID] = int(Dad.worldPosition.y*100)
    internet.var.PosZ[internet.var.ID] = int(Dad.worldPosition.z*100)

    internet.var.RotX[internet.var.ID] = int(BubOri.x*100)
    internet.var.RotY[internet.var.ID] = int(BubOri.y*100)
    internet.var.RotZ[internet.var.ID] = int(BubOri.z*100)

    own["ID"] = "ID:" + str(internet.var.ID) + " X:" + str(internet.var.PosX[internet.var.ID])
    #Variables globales
if gl.OnlineMode == True:
    internet.players = 1
    gl.OnlineOBJ = [0]
    gl.XarxaON = True
    gl.OnlineMode = False
```

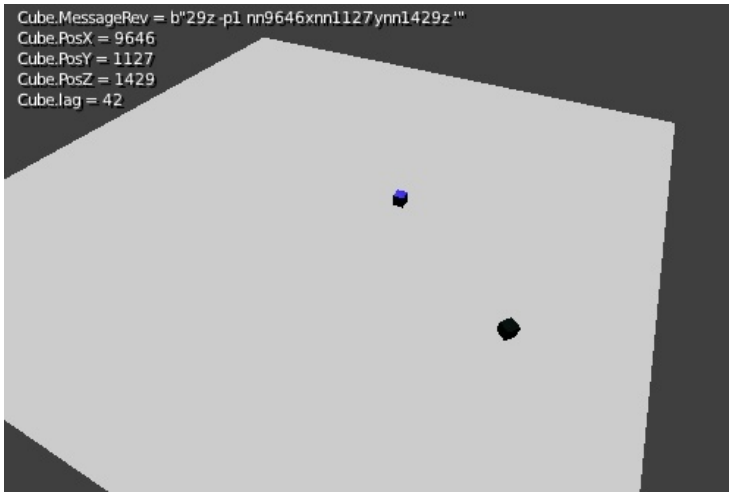
Aquest codi és força entenedor en general. Només destacar que per enviar les dades de rotació és apropiat fer-ho amb la funció *.to_euler()*, encarregada de llegir la matriu tridimensional de la rotació existent en Blender.

Entre altres coses enviem les dades numèriques com a variables *integer*. Per poder tenir un valor més exacte multipliquem aquest valor per 100 abans de ser enviat.

5.2.5 Test final i de rendiment.

Un cop fet l'script podem crear un petit escenari en el que provar l'script i comprovar-ne el rendiment. En primer lloc codifiquem les dades a enviar per poder-ne obtenir els valors de les coordenades corresponents a l'objecte remot. També podem obtenir el numero de dades per segon que rebem, tan tenint en compte els *bytes* per segon, com les coordenades per segon.

A la imatge de la dreta es mostra un moment del programa de test. La primera variable, propietat de l'objecte *cube* ens mostre el missatge que estem revent en aquest moment, el meteix que s'esta processant. Aquest missatge es compon del indicador *-p* que fa referencia al jugador. Les lletres *n*, són caràcters nuls, que s'eliminaran i serveixen únicament per mantenir 6 dígits per cada coordenada. Les lletres x, y i z ens indiquen la coordenada a la que correspon el valor anterior.



Finalment la propietat “lag” ens indica la velocitat en que rebem les coordenades, basant-se en la coordenada x. Cal dir que aquesta velocitat no és la real, dons en aquesta versió del mode en xarxa hi ha alguns errors que fan que la la velocitat real sigui pèssima. Actualment s'ha millorat enormement i el problema del *lag* a quedat solucionat.

6. No és el final.

6.1 La web del projecte.

Com a complement del projecte i per obrir aquest a les portes d'Internet s'ha creat una web des de la qual es podrà descarregar gratuïtament el videojoc i s'informara de noves actualitzacions i millores. La web estarà disponible a partir de la presentació d'aquest dossier a la direcció: http://bubbleracer.xtrweb.com/ o a la direcció http://www.bubbleracer.free si el domini *free* es converteix en realitat algun dia.

La web, de la mateixa manera que la resta del projecte, ha estat realitzada únicament amb software lliure i utilitzant exclusivament contingut propi. El software utilitzat correspon a: Notepad++, Gimp, FileZilla, Gedit i fent ús dels llenguatges HTML4, PHP i CSS.

6.3 Conclusions

Com ja temia des del començament del projecte, aquest és un projecte sense final. Per molt que avanci o desenvolupi millores o noves funcions, per molts circuits, models o cançons que pugui afegir, sempre es podrà millorar. Però això no em desanima, sinó que em motiva més. Saber que cada cop pot ser millor i millor, sense que existeixi cap tipus de limit ni restricció.

Aquest és un dels tants exemples de les enormes possibilitats de Blender, i que, tot hi els canvis de la versió 2.6 i l'aparició de nous motors de joc moderns, Blender és una alternativa a tenir en compte també com a Game Engine.

Personalment crec que Blender te algunes mancances importants com a motor de joc, però malgrat tot segueixo creient que és un dels millors motors lliures. Per solucionar aquestes mancances alguns usuaris han creat GameKit, un projecte de *game engine* pensat per funcionar amb Blender.

No ha estat un projecte fàcil. He agut de crear el meu propi material, textures, gràfics i models, el que significa un munt d'hores invertides en la seva creació. A això li tenim que afegir les hores invertides en programació. He hagut de buscar músiques i fonts, vigilant sempre que no fos material amb copyright, lo que descartava gran part de les fonts i gairebé el total de les bandes sonores que trobava. Pel que fa la recerca, ha suposat haver d'aprendre de zero un nou llenguatge de programació, Python 3 (i la llibreria BGE), sense cap experiència prèvia i amb la gran majoria d'informació en Anglès.

Actualment segueixo millorant i traient noves versions del joc. Malgrat que el videojoc durant la redacció del treball es trobava en la seva versió 0.30, ara ja treballo sobre la versió 0.33 on s'ha integrat el mode en xarxa i s'han afegit moltíssimes millores.

6.4 Webgrafia.

SO		
http://www.ubuntu.cat/	Ubuntu 12.10	Una del les distribucions més famoses de Linux.
Software		
http://www.blender.org/ (Anglés)	Blender 2.4.9b i Blender 2.6.1	Creació de models 3D i Motor de Joc
http://www.python.org/ (Anglés)	Python 3	Programació per al videojoc.
http://www.gimp.org/	Gimp 2.6.11	Gràfics i retoc d'imatge.
http://audacity.sourceforge.net/	Audacity 1.3 Beta + LAME 3.98.3	Retoc bàsic de efectes de so i conversió.
http://www.videolan.org/vlc/	Videolan 2.0.1	Reproductor i convertidor de vídeo.
http://www.softcatala.org/	Web amb el software anterior en català, excepte el marcat com Anglés.	
Documentació		
http://wiki.blender.org/index.php/CA/Main_Page	La documentació oficial de Blender. Conte, majoritàriament en anglés, informació sobre varies versions de Blender, sobre Python, tutorials bàsics i avançats, etc.	
http://www.tutorialsforblender3d.com/ (Anglés)	Documentació de Python i la llibreria BGE per a Blender 2.4, però útil igualment en la majoria dels casos.	
http://docs.python.org/ (Anglés)	Documentació oficial de Python. Inclou ajuda per la instal·lació i utilització d'aquest, tal com l'aplicació 2to3 que converteix codi en Python 2.7 a Python 3.	
Comunitats		
http://www.foro3d.com/foro3d.php (Castellà)	La major comunitat de parla hispana sobre software i creació 3D.	
http://www.g-blender.org/ (Castellà)	Comunitat hispana exclusivament de Blender.	
http://fisicomolon.blogspot.com.es/ (Castellà)	Noticies, videotutorials i un foro fan d'aquesta web una de les més visitades per a aprenents de Blender Game Engine.	
http://www.niel3d.com (Castellà)	Fòrum de parla hispana per als usuaris de Blender.	
http://blenderartists.org (Anglés)	Una de les comunitats de Blender més grans del món.	
Webs de projectes i usuaris		
http://mikepan.com/ (Anglés)	Web personal d'en Mike, un professional de Blender que posa a la nostra disposició les seves creacions.	
http://solarlune-gameup.blogspot.com.es/ (Anglés)	Un altre web amb projectes i tutorials, on es crea un videjoc amb Blender, encara que utilitzen molt els Logic Bricks i poc Python.	
http://www.blendenzo.com/ (Anglés)	Usuari de Blender Game Engine, proporciona tutorials i altres.	
Material		
http://3dmodelsheets.com/ (Anglés)	Textures repetitives i gratuïtes per al nostre videojoc.	
http://cgcookie.com/blender/2010/02/12/textures-skies-part-deux/ (Anglés)	Pack de textures panoràmiques de cels, sota llicencia Creative Commons 3.0 Attribuition Wes Burke.	
http://www.fontsquirrel.com/ (Anglés)	Fonts de lletra gratuïtes i de llicencia per a ús comercial.	
http://dig.ccmixter.org/dig?dig-lic=safe (Anglés)	Músiques gratuïtes i amb llicencia per a ús comercial.	
http://soundcloud.com/ (Anglés)	Pàgina per compartir música. Algunes cançons tenen llicencies lliures.	

http://www.youtube.com/user/TheMusicRack (Anglés)	Canal de youtube amb músiques gratuïtes de tot tipus.
http://lmms.sourceforge.net/ (Anglés)	Programa de creació MIDI lliure i de codi obert. Compte amb un apartat per compartir les teves creacions i descarregar o millorar les de altres usuaris.
http://freemusicarchive.org/music/Kevin_MacLeod/Funk_Sampler/There_It_Is_ (Anglés)	Música amb llicencies gratuïtes, algunes de Kevin McLeod han estat usades en el projecte.

Altres
http://victorpando.blogspot.com.es/2008/12/programacin-de-sockets-con-python.html (Castellà)
http://www.inf-cr.uclm.es/www/cglez/ (Castellà)
http://www.rozengain.com/blog/2009/06/11/beginners-tutorial-using-video-textures-in-the-blender-game-engine/ (Anglés)
http://www.fisicomolon.com/joomla/index.php?option=com_content&view=article&id=52:tutorial-no-8-manejar-texto&catid=36:blender-24x&Itemid=55 (Castellà)
http://wiki.blender.org/index.php/Dev:Ref/Release_Notes/2.62/Game_Engine (Anglés)
https://www.youtube.com/user/goranmilovano (Anglés)
Molt interessant: http://geta3d.com/ (Anglés)
Molt interessant: https://www.youtube.com/watch?v=JVB1ayT6Fdc (Anglés)
Imatges del treball extretes d'Internet Ordenades per ordre de aparició
http://innovate.ucsb.edu/799-richard-stallman-free-software-and-copyleft (Anglés)
http://games.softpedia.com/progScreenshots/Blitz3D-Demo-Screenshot-36867.html (Anglés)
http://www.cgchannel.com/2010/02/new-version-of-udk/ (Anglés)
http://en.wikipedia.org/wiki/Low_poly (Anglés)

7. Annexos

Bubble Racer v0.3x

A. Imatges del joc


0:38.00
0:14.31
Ultim temps: 00:00.00
Millor temps: 00:00.00
63 km/h
IP Local: 192.168.1.15
IP Publica: 85.50.238.92
Servidor
Tipos
Usuaris
NAT IPv4 TEST
TEST
0/0
Localhost
TEST
1/2
BR Server
TEST
0/14
Connectar per IP: 192.168.
CREAR SERVIDOR
JUGAR
EN XARXA
OPCIONS
ROBERT PLANAS
INS NARCÍS MONTURIOL
SORTIR

B. Codi font: Bubble Racer v0.31

Fitxer: Arranque

```
import bge
import os
gl = bge.logic

#Configuracions inicials. Resolucio.
try:
    file = open('conf.txt')
    config = file.readlines()

    con0 = config[0].replace('ResX=','')
    con1 = config[1].replace('ResY=','')
    gl.globalDict["ResX"] = con0.replace('\n','')
    gl.globalDict["ResY"] = con1.replace('\n','')
    VS = config[2].replace('\n','')
    VM = config[3]
    gl.globalDict["VolumMusica"] = float(VM)
    gl.globalDict["VolumSo"] = float(VS)
    file.close()

bge.render.setWindowSize(int(gl.globalDict["ResX"]),int(gl.globalDict["ResY"]))
except:
    print("No s'ha trobat l'arxiu conf.txt")
    gl.globalDict["VolumMusica"] = 1
    gl.globalDict["VolumSo"] = 0.8

#Declaracio de globals
gl.globalDict["Loading"] = False
gl.globalDict["meta1"] = False
gl.globalDict["meta2"] = False
gl.globalDict["meta3"] = False
gl.globalDict["meta4"] = False
gl.globalDict["UltimoTiempo"] = '00:00.00'
gl.globalDict["MejorTiempo"] = 0
gl.globalDict["MejorTiempo2"] = 99999
gl.globalDict["Incl.Sumatori"] = 0
gl.globalDict["Incl.Max"] = 0.0000
gl.globalDict["Incl.Max2"] = 0
gl.globalDict["MenuPos"] = 1
gl.globalDict["PreMenuPos"] = 1
gl.globalDict["MenuLabel"] = 1
gl.globalDict["MenuEnd"] = 4
gl.globalDict["SelectPos"] = 1
gl.globalDict["PreSelectPos"] = 1
gl.globalDict["SelectLabel"] = 0
gl.globalDict["SelectEnd"] = 8
gl.globalDict["Bubble"] = 1
gl.globalDict["Start"] = False
gl.globalDict["GameMenu"] = 1
gl.globalDict["GameMenuAct"] = False
gl.globalDict["MenuStart"] = False
gl.items = 0
```

Fitxer: IniVideo

```
import bge
import GameLogic
import os

controller = bge.logic.getCurrentController()
scene = bge.logic.getCurrentScene()

obj = controller.owner

# Comprova que el video existeix.
if "Video" in obj:
        video = obj["Video"]# Va a l'objecte video i el refresca.
        video.refresh(True)

else:
    #Crea l'objecte video (Dins de l'escena actual com si fos un material)
        matID = bge.texture.materialID(obj, "MA" + "Pantalla")
        video = bge.texture.Texture(obj, matID)
        movie = bge.logic.expandPath('//title.avi')

        video.source = bge.texture.VideoFFmpeg(movie)

        video.source.scale = True # Escala el video
        obj["Video"] = video #Crea l'objecte video.

        # check for optional loop property
if "loop" in obj: # loop it forever
    if obj['loop'] == True:
        video.source.repeat = -1 # no looping
    else:
        video.source.repeat = 0

# start the video
video.source.play()

if obj["Temps"] > 2:
    scene.replace("Intro")
```

Fitxer: Menu

```
import bge
import GameLogic
import os
import internet

controller = bge.logic.getCurrentController()
keyboard = bge.logic.keyboard
JUST_ACTIVATED = bge.logic.KX_INPUT_JUST_ACTIVATED
scene = bge.logic.getCurrentScene()

JugarOBJ = scene.objects["JUGAR"]
XarxaOBJ = scene.objects["EN_XARXA"]
OpcionsOBJ = scene.objects["OPCIONS"]
SortirOBJ = scene.objects["SORTIR"]
TextOBJ = scene.objects["Text"]
Text2OBJ = scene.objects["Text2"]
Text3OBJ = scene.objects["Text3"]
Text4OBJ = scene.objects["Text4"]
Text5OBJ = scene.objects["Text5"]
OSelectorOBJ = scene.objects["OnlineSelector"]
OLightOBJ = scene.objects["OnlineLight"]
CSLightOBJ = scene.objects["CREARSERVIDORLIGHT"]
SelectorOBJ = scene.objects["Selector"]
LlumOBJ = scene.objects["Llumi"]
CamOBJ = scene.objects["Camera"]
EmptyOBJ = scene.objects["Empty"]
ResTXT = scene.objects["Res"]
TbubTXT = scene.objects["OPBUB"]
TopTXT = scene.objects["Res.002"]
MusicaTXT = scene.objects["MUSICA"]
SoTXT = scene.objects["SOTXT"]

sound = controller.actuators["Sound"]

JSelector = controller.actuators["Jmenu_selector"]
controller.activate(JSelector)
JSelector.dLoc = [0,0,0]

obj = controller.owner


#Configuracions del menu
------------------------------------------------------------
gl = bge.logic
controller.activate(sound)

if gl.globalDict["Loading"] == True:
    scene.replace("Game")

try:
    sound.volume = gl.globalDict["VolumMusica"]
except:
    print("Error: No s'ha pogut actualitzar el Volum")

if gl.globalDict["MenuStart"] == False: #Previes
    try:
        file = open('conf.txt')
        config = file.readlines()
        Config1 = config[0].replace('ResX=','')
        Config2 = config[1].replace('ResY=','')
        All = ResTXT["Text"] = Config1 + 'x' + Config2
        ResTXT["Text"] = All.replace('\n','')
        musica.value = '"' + str(gl.globalDict["VolumMusica"]*100) + '%' + '"'
        so.value = '"' + str(gl.globalDict["VolumSo"]*100) + '%' + '"'
    except:
        ResTXT["Text"] = str(720) + 'x' + str(480)

def OnlineIPs():
    TextOBJ.resolution = 6
    TextOBJ.size = 0.78
    TextOBJ.color = [1,1.0,1.0,0.5]
    Text2OBJ.resolution = 6
    Text2OBJ.size = 0.78
    Text2OBJ.color = [1,1.0,1.0,0.5]
    Text3OBJ.resolution = 6
    Text3OBJ.size = 0.78
    Text3OBJ.color = [1,1.0,1.0,0.3]
    Text4OBJ.resolution = 6
    Text4OBJ.size = 0.78
    Text4OBJ.color = [1,1.0,1.0,0.3]
    Text5OBJ.resolution = 6
    Text5OBJ.size = 0.78
    Text5OBJ.color = [1,1.0,1.0,0.3]

    Text4OBJ.text = ""
    Text5OBJ.text = ""
    Text3OBJ.text = "Cercant..."
    TextOBJ.text = str(internet.IPlocal())
    Text2OBJ.text = str(internet.IPpublica())

    gl.items = 0

def OnlineSL():
    if Text3OBJ["Actual"] >= 5:
        if Text2OBJ.text == "Error":
            Text3OBJ.text = "Error de connexio"
        else:
            internet.tmp = 0
            ServerList = internet.ServerList()
            Text3OBJ.text = internet.MakeList(ServerList)
             internet.tmp = 2
             Text4OBJ.text = internet.MakeList(ServerList)
             internet.tmp = 3
             Text5OBJ.text = internet.MakeList(ServerList)

             gl.items = int(len(ServerList) / 4)

             Text3OBJ["Actual"] = 0

gl.globalDict["MenuStart"] = True
if gl.globalDict["MenuStart"] == True: #Temps del video d'intro, recomenat: 13
    obj.position.z = -0.2

```

```
    #Opcions de moviment amb les fletxes.
    if keyboard.events[bge.events.DOWNARROWKEY] == JUST_ACTIVATED:
        gl.globalDict["MenuPos"] = gl.globalDict["MenuPos"] + 1
        if gl.globalDict["MenuPos"] == gl.globalDict["MenuEnd"] + 1:
            gl.globalDict["MenuPos"] = 1

    if keyboard.events[bge.events.UPARROWKEY] == JUST_ACTIVATED:
        gl.globalDict["MenuPos"] = gl.globalDict["MenuPos"] - 1
        if gl.globalDict["MenuPos"] == 0:
            gl.globalDict["MenuPos"] = gl.globalDict["MenuEnd"]

    if keyboard.events[bge.events.RIGHTARROWKEY] == JUST_ACTIVATED:
        gl.globalDict["SelectPos"] = gl.globalDict["SelectPos"] + 1
        if gl.globalDict["SelectPos"] == gl.globalDict["SelectEnd"] + 1:
            gl.globalDict["SelectPos"] = 1

    if keyboard.events[bge.events.LEFTARROWKEY] == JUST_ACTIVATED:
        gl.globalDict["SelectPos"] = gl.globalDict["SelectPos"] - 1
        if gl.globalDict["SelectPos"] == 0:
            gl.globalDict["SelectPos"] = gl.globalDict["SelectEnd"]


    if gl.globalDict["MenuLabel"] == 1: #Menu d'inici.

        gl.globalDict["MenuEnd"] = 4
        #Animacions dels botons.
        if keyboard.events[bge.events.DOWNARROWKEY] == JUST_ACTIVATED or \
        keyboard.events[bge.events.UPARROWKEY] == JUST_ACTIVATED:
            if gl.globalDict["MenuPos"] == 1:
                JugarOBJ.localPosition.x = -0.9
                XarxaOBJ.localPosition.x = -1.1
                OpcionsOBJ.localPosition.x = -1.1
                SortirOBJ.localPosition.x = 1.5
            if gl.globalDict["MenuPos"] == 2:
                JugarOBJ.localPosition.x = -1.1
                XarxaOBJ.localPosition.x = -0.9
                OpcionsOBJ.localPosition.x = -1.1
            if gl.globalDict["MenuPos"] == 3:
                JugarOBJ.localPosition.x = -1.1
                XarxaOBJ.localPosition.x = -1.1
                OpcionsOBJ.localPosition.x = -0.9
                SortirOBJ.localPosition.x = 1.5
            if gl.globalDict["MenuPos"] == 4:
                JugarOBJ.localPosition.x = -1.1
                OpcionsOBJ.localPosition.x = -1.1
                SortirOBJ.localPosition.x = 1.3
            gl.globalDict["PreMenuPos"] = gl.globalDict["MenuPos"]

        #Accions dels botons.
        if keyboard.events[bge.events.RETKEY] == JUST_ACTIVATED:
            if gl.globalDict["MenuPos"] == 1:
                gl.globalDict["MenuLabel"] = 3
                gl.globalDict["PreMenuPos"] = 1
                gl.globalDict["MenuPos"] = 1
                EmptyOBJ.localPosition.y = 0
                gl.globalDict["SelectLabel"] = 1
                gl.globalDict["MenuEnd"] = 2

            if gl.globalDict["MenuPos"] == 2:
                gl.globalDict["MenuLabel"] = 4
                gl.globalDict["PreMenuPos"] = 1
                gl.globalDict["MenuPos"] = 1
                CamOBJ.localPosition.x = -6
                gl.globalDict["SelectLabel"] = 1
                gl.globalDict["MenuEnd"] = 1

                Text3OBJ["Actual"] = 4
                OnlineIPs()


            if gl.globalDict["MenuPos"] == 3:
                gl.globalDict["MenuLabel"] = 2
                gl.globalDict["PreMenuPos"] = 1
                gl.globalDict["MenuPos"] = 1
                CamOBJ.localPosition.x = 3.6
                gl.globalDict["SelectLabel"] = 1
                gl.globalDict["MenuEnd"] = 5

            if gl.globalDict["MenuPos"] == 4:
                gl.endGame()

    if gl.globalDict["MenuLabel"] == 2: #Menu d'opcions.

        if keyboard.events[bge.events.ESCKEY] == JUST_ACTIVATED:
            gl.globalDict["MenuLabel"] = 1
            gl.globalDict["PreMenuPos"] = 3
            gl.globalDict["MenuPos"] = 3
            CamOBJ.localPosition.x = 0
            LlumOBJ.localPosition.y = -3
            SelectorOBJ.localPosition.y = 0.5

        if keyboard.events[bge.events.DOWNARROWKEY] == JUST_ACTIVATED or \
        keyboard.events[bge.events.UPARROWKEY] == JUST_ACTIVATED:

            if gl.globalDict["MenuPos"] == 1:
                SelectorOBJ.localPosition.y = 0.5
                LlumOBJ.localPosition.y = -3
            if gl.globalDict["MenuPos"] == 2:
                SelectorOBJ.localPosition.y = 0.17
            if gl.globalDict["MenuPos"] == 3:
                SelectorOBJ.localPosition.y = -0.16
                LlumOBJ.localPosition.y = -3
            if gl.globalDict["MenuPos"] == 4:
                LlumOBJ.localPosition.y = -0.34
                SelectorOBJ.localPosition.y = -3
            if gl.globalDict["MenuPos"] == 5:
                LlumOBJ.localPosition.y = -0.67
                SelectorOBJ.localPosition.y = -3

            gl.globalDict["SelectPos"] = 1
            gl.globalDict["PreMenuPos"] = gl.globalDict["MenuPos"]
```

```
224.
225.         if gl.globalDict["MenuPos"] == 1: #Resolucio
226.
227.             if keyboard.events[bge.events.LEFTARROWKEY] ==
    JUST_ACTIVATED or \
228.             keyboard.events[bge.events.RIGHTARROWKEY] ==
    JUST_ACTIVATED:
229.
230.                 if gl.globalDict["SelectPos"] == 1:
231.                     ResX = 720
232.                     ResY = 480
233.                 if gl.globalDict["SelectPos"] == 2:
234.                     ResX = 800
235.                     ResY = 600
236.                 if gl.globalDict["SelectPos"] == 3:
237.                     ResX = 1024
238.                     ResY = 720
239.                 if gl.globalDict["SelectPos"] == 4:
240.                     ResX = 1024
241.                     ResY = 768
242.                 if gl.globalDict["SelectPos"] == 5:
243.                     ResX = 1280
244.                     ResY = 720
245.                 if gl.globalDict["SelectPos"] == 6:
246.                     ResX = 1280
247.                     ResY = 768
248.                 if gl.globalDict["SelectPos"] == 7:
249.                     ResX = 1360
250.                     ResY = 768
251.                 if gl.globalDict["SelectPos"] == 8:
252.                     ResX = 1366
253.                     ResY = 768
254.
255.                 ResTXT["Text"] = str(ResX) + 'x' + str(ResY)
256.                 gl.globalDict["ResX"] = str(ResX)
257.                 gl.globalDict["ResY"] = str(ResY)
258.                 gl.globalDict["PreSelectPos"] =
    gl.globalDict["SelectPos"]
259.                 gl.globalDict["SelectEnd"] = 8
260.
261.         if gl.globalDict["MenuPos"] == 2: #Musica
262.
263.             if keyboard.events[bge.events.LEFTARROWKEY] ==
    JUST_ACTIVATED or \
264.             keyboard.events[bge.events.RIGHTARROWKEY] ==
    JUST_ACTIVATED:
265.
266.                 if gl.globalDict["SelectPos"] == 1:
267.                     muvol = 0
268.                 if gl.globalDict["SelectPos"] == 2:
269.                     muvol = 10
270.                 if gl.globalDict["SelectPos"] == 3:
271.                     muvol = 20
272.                 if gl.globalDict["SelectPos"] == 4:
273.                     muvol = 30
274.                 if gl.globalDict["SelectPos"] == 5:
275.                     muvol = 40
276.                 if gl.globalDict["SelectPos"] == 6:
277.                     muvol = 50
278.                 if gl.globalDict["SelectPos"] == 7:
279.                     muvol = 60
280.                 if gl.globalDict["SelectPos"] == 8:
281.                     muvol = 70
282.                 if gl.globalDict["SelectPos"] == 9:
283.                     muvol = 80
284.                 if gl.globalDict["SelectPos"] == 10:
285.                     muvol = 90
286.                 if gl.globalDict["SelectPos"] == 11:
287.                     muvol = 100
288.
289.                 MusicaTXT["Text"] = str(muvol) + "%"
290.                 gl.globalDict["SelectEnd"] = 11
291.                 gl.globalDict["PreSelectPos"] =
    gl.globalDict["SelectPos"]
292.                 gl.globalDict["VolumMusica"] =
    (gl.globalDict["SelectPos"] - 1) / 10
293.
294.         if gl.globalDict["MenuPos"] == 3: #So
295.
296.             if keyboard.events[bge.events.LEFTARROWKEY] ==
    JUST_ACTIVATED or \
297.             keyboard.events[bge.events.RIGHTARROWKEY] ==
    JUST_ACTIVATED:
298.
299.                 if gl.globalDict["SelectPos"] == 1:
300.                     sovol = 0
301.                 if gl.globalDict["SelectPos"] == 2:
302.                     sovol = 10
303.                 if gl.globalDict["SelectPos"] == 3:
304.                     sovol = 20
305.                 if gl.globalDict["SelectPos"] == 4:
306.                     sovol = 30
307.                 if gl.globalDict["SelectPos"] == 5:
308.                     sovol = 40
309.                 if gl.globalDict["SelectPos"] == 6:
310.                     sovol = 50
311.                 if gl.globalDict["SelectPos"] == 7:
312.                     sovol = 60
313.                 if gl.globalDict["SelectPos"] == 8:
314.                     sovol = 70
315.                 if gl.globalDict["SelectPos"] == 9:
316.                     sovol = 80
317.                 if gl.globalDict["SelectPos"] == 10:
318.                     sovol = 90
319.                 if gl.globalDict["SelectPos"] == 11:
320.                     sovol = 100
321.
322.                 SoTXT["Text"] = str(sovol) + "%"
323.                 gl.globalDict["PreSelectPos"] =
    gl.globalDict["SelectPos"]
324.                 gl.globalDict["VolumSo"] =
    (gl.globalDict["SelectPos"] - 1) / 10
325.                 gl.globalDict["SelectEnd"] = 11
326.
327.         if gl.globalDict["MenuPos"] == 4:
328.
329.             if keyboard.events[bge.events.RETKEY] ==
    JUST_ACTIVATED:
330.
331.
332.
    bge.render.setWindowSize(int(gl.globalDict["ResX"]),
    int(gl.globalDict["ResY"]))
333.
334.
335.                 file2 = open('conf.txt', 'w')
336.
337.                 file2text = 'ResX=' + gl.globalDict["ResX"] +
    '\n' + "ResY=" + gl.globalDict["ResY"] + \
338.                 '\n' + str(gl.globalDict["VolumMusica"]) +
    '\n' + str(gl.globalDict["VolumSo"])
339.                 file2.write(file2text)
340.                 file2.close()
```

```
341.
342.
343.         if gl.globalDict["MenuPos"] == 5:
344.
345.             if keyboard.events[bge.events.RETKEY] ==
    JUST_ACTIVATED:
346.                 gl.globalDict["MenuLabel"] = 1
347.                 gl.globalDict["PreMenuPos"] = 3
348.                 gl.globalDict["MenuPos"] = 3
349.                 CamOBJ.localPosition.x = 0
350.                 LlumOBJ.localPosition.y = -3
351.                 SelectorOBJ.localPosition.y = 0.5
352.
353.     if gl.globalDict["MenuLabel"] == 3: #Menu de jugar.
354.
355.         if keyboard.events[bge.events.ESCKEY] ==
    JUST_ACTIVATED:
356.             if gl.globalDict["MenuPos"] == 2:
357.                 JSelector.dLoc = [0,0.3,0]
358.             gl.globalDict["MenuLabel"] = 1
359.             gl.globalDict["PreMenuPos"] = 1
360.             gl.globalDict["MenuPos"] = 1
361.             EmptyOBJ.localPosition.y = 2
362.
363.         if keyboard.events[bge.events.DOWNARROWKEY] ==
    JUST_ACTIVATED or \
364.         keyboard.events[bge.events.UPARROWKEY] ==
    JUST_ACTIVATED:
365.
366.             if gl.globalDict["MenuPos"] == 1:
367.                 JSelector.dLoc = [0,0.3,0]
368.             if gl.globalDict["MenuPos"] == 2:
369.                 JSelector.dLoc = [0,-0.3,0]
370.
371.             gl.globalDict["SelectPos"] = 1
372.             gl.globalDict["PreMenuPos"] =
    gl.globalDict["MenuPos"]
373.
374.         if gl.globalDict["MenuPos"] == 1:
375.
376.             if keyboard.events[bge.events.LEFTARROWKEY] ==
    JUST_ACTIVATED or \
377.             keyboard.events[bge.events.RIGHTARROWKEY] ==
    JUST_ACTIVATED:
378.
379.                 if gl.globalDict["SelectPos"] == 1:
380.                     TbubTXT["Text"] = '   Bubble 1'
381.                 if gl.globalDict["SelectPos"] == 2:
382.                     TbubTXT["Text"] = '   Bubble 2'
383.                 if gl.globalDict["SelectPos"] == 3:
384.                     TbubTXT["Text"] = '   Bubble 3'
385.
386.                 gl.globalDict["Bubble"] =
    gl.globalDict["SelectPos"]
387.                 gl.globalDict["PreSelectPos"] =
    gl.globalDict["SelectPos"]
388.                 gl.globalDict["SelectEnd"] = 3
389.
390.         if gl.globalDict["MenuPos"] == 2:
391.
392.                 if gl.globalDict["SelectPos"] == 1:
393.                     TopTXT["Text"] = '        Jugar'
394.                 if keyboard.events[bge.events.RETKEY] ==
    JUST_ACTIVATED:
395.                     CamOBJ.localPosition.y = 3
396.                     gl.globalDict["Loading"] = True
397.
398.                 if gl.globalDict["SelectPos"] == 2:
399.                     TopTXT["Text"] = ' Dos jugadors'
400.                 if gl.globalDict["SelectPos"] == 3:
401.                     TopTXT["Text"] = '      Tornar'
402.                     if keyboard.events[bge.events.RETKEY] ==
    JUST_ACTIVATED:
403.                         gl.globalDict["MenuLabel"] = 1
404.                         gl.globalDict["PreMenuPos"] = 1
405.                         gl.globalDict["MenuPos"] = 1
406.                         EmptyOBJ.localPosition.y = 2
407.                         JSelector.dLoc = [0,0.3,0]
408.
409.                 gl.globalDict["PreSelectPos"] =
    gl.globalDict["SelectPos"]
410.                 gl.globalDict["SelectEnd"] = 3
411.
412.
413.     if gl.globalDict["MenuLabel"] == 4: #Menu En xarxa
    (GENERAL)
414.         gl.globalDict["MenuEnd"] = 2
415.
416.         OnlineSL()
417.         OSelectorOBJ.visible = False
418.
419.         if keyboard.events[bge.events.ESCKEY] ==
    JUST_ACTIVATED:
420.             gl.globalDict["MenuLabel"] = 1
421.             gl.globalDict["PreMenuPos"] = 2
422.             gl.globalDict["MenuPos"] = 2
423.             CamOBJ.localPosition.x = 0
424.
425.
426.         if keyboard.events[bge.events.DOWNARROWKEY] ==
    JUST_ACTIVATED or \
427.         keyboard.events[bge.events.UPARROWKEY] ==
    JUST_ACTIVATED:
428.
429.             if gl.globalDict["MenuPos"] == 1:
430.                 OLightOBJ.visible = True
431.                 CSLightOBJ.localPosition.y = -1.15
432.             if gl.globalDict["MenuPos"] == 2:
433.                 OLightOBJ.visible = False
434.                 CSLightOBJ.localPosition.y = -0.65
435.
436.         if keyboard.events[bge.events.RETKEY] ==
    JUST_ACTIVATED:
437.             if gl.globalDict["MenuPos"] == 1:
438.                 if gl.items > 0:
439.                         gl.globalDict["MenuLabel"] = 5
440.                         gl.globalDict["MenuPos"] =
    gl.globalDict["PreMenuPos"]
441.             if gl.globalDict["MenuPos"] == 2:
442.                 gl.globalDict["MenuLabel"] = 6
443.                 CamOBJ.localPosition.y = -2.4
444.
```

```
445.
446.     if gl.globalDict["MenuLabel"] == 5: #Seleccio de
    servidor
447.
448.         OnlineSL()
449.         OLightOBJ.visible = False
450.         gl.globalDict["MenuEnd"] = gl.items
451.         if gl.items < gl.globalDict["MenuPos"]:
452.             gl.globalDict["MenuPos"] = gl.items
453.             OSelectorOBJ.localPosition.y = 0.433 -
    0.053*(gl.globalDict["MenuPos"] - 1)
454.
455.         if gl.globalDict["MenuEnd"] > 0:
456.             OSelectorOBJ.visible = True
457.
458.         if keyboard.events[bge.events.ESCKEY] ==
    JUST_ACTIVATED:
459.             gl.globalDict["MenuLabel"] = 4
460.             gl.globalDict["MenuPos"] = 1
461.             OLightOBJ.visible = True
462.
463.         if keyboard.events[bge.events.RETKEY] ==
    JUST_ACTIVATED:
464.             internet.tmp = 1
465.             IPs =
    internet.MakeList(internet.ServerList()).split('\n')
466.             internet.IP = IPs[gl.globalDict["MenuPos"] - 1]
467.             internet.Connect()
468.
469.         if keyboard.events[bge.events.DOWNARROWKEY] ==
    JUST_ACTIVATED or \
470.         keyboard.events[bge.events.UPARROWKEY] ==
    JUST_ACTIVATED:
471.
472.             OSelectorOBJ.localPosition.y = 0.433 -
    0.053*(gl.globalDict["MenuPos"] - 1)
473.
474.             gl.globalDict["SelectPos"] = 1
475.             gl.globalDict["PreMenuPos"] =
    gl.globalDict["MenuPos"]
476.
477.     if gl.globalDict["MenuLabel"] == 6:
478.
479.         if keyboard.events[bge.events.ESCKEY] ==
    JUST_ACTIVATED:
480.             gl.globalDict["MenuLabel"] = 4
481.             gl.globalDict["MenuPos"] = 2
482.             CamOBJ.localPosition.y = 0
```

Fitxer: internet.py

```
1. import socket
2. import urllib.request
3.
4. def IPlocal():
5.     return socket.gethostbyname(socket.gethostname())
6.
7. def IPpublica():
8.     WebInfo =
    urllib.request.urlopen("http://bubbleracer.xtrweb.com/res/ipi
    nfo.php")
9.     IPp = WebInfo.read().decode("utf8")
10.     WebInfo.close()
11.     return IPp
12.
13. def ServerList():
14.     WebServer =
    urllib.request.urlopen("http://bubbleracer.xtrweb.com/res/ser
    verlist.txt")
15.     ServerText = WebServer.read().decode("utf8")
16.     WebServer.close()
17.     return ServerText.split('\n')
18.
19. def MakeList(List):
20.     Lfinal = ""
21.     for a in range(tmp,len(List),4):
22.         Lfinal = Lfinal + "\n" + List[a]
23.     return Lfinal[1:]
24.
25. def Connect():
26.     client = socket.socket(socket.AF_INET,
    socket.SOCK_STREAM)
27.     client.connect((str(IP), 6080))
28.     print("Change YES!")
```

```
    print1["Text"] = ProcesarTemps(own["timer"])

    if gl.globalDict["meta1"] == True and \
    gl.globalDict["meta2"] == True and \
    gl.globalDict["meta3"] == True and\
    gl.globalDict["meta4"] == True:
        if own["timer"] > 1:
            gl.globalDict["UltimoTiempo"] = print1["Text"]
            gl.globalDict["MejorTiempo"] = own["timer"]
             if gl.globalDict["MejorTiempo"] < gl.globalDict["MejorTiempo2"]:
                 gl.globalDict["MejorTiempo2"] = gl.globalDict["MejorTiempo"]

         own["timer"] = 0
         gl.globalDict["meta1"] = False
         gl.globalDict["meta2"] = False
         gl.globalDict["meta3"] = False
         gl.globalDict["meta4"] = False

    if gl.globalDict["meta1"] == True and gl.globalDict["meta2"] == True:
         gl.globalDict["meta1"] = False

    #Start = True
    if own["timer"] > 0 and MenuBack.visible == False:
        gl.globalDict["Start"] = True

    #Velocitat
    #acc3 = gl.globalDict["acc2"]
    acc3 = Servo.linV[1]
    if gl.globalDict["Bubble"] == 1:
        vel = int(acc3*150)
    if gl.globalDict["Bubble"] == 2:
        vel = int(acc3*180)
    if gl.globalDict["Bubble"] == 3:
        vel = int(acc3*180)

    print2["Text"] = str(vel) + " km/h"

    #Ultim temps
    print3["Text"] = gl.globalDict["UltimoTiempo"]

    #Millor temps
    print4["Text"] = ProcesarTemps(gl.globalDict["MejorTiempo2"])
    if gl.globalDict["MejorTiempo2"] == 99999:
        print4["Text"] = '00:00.00'

def ProcesarTemps(temps):
    minuts = int(temps/60)
    segons = int(temps - minuts*60)
    decimes = int((temps - segons - minuts*60) * 100)
    temps = str(minuts) + ":" + str("%02i"% segons) + "." + str(decimes)
    return temps

def PauseMenu():

    cont = bge.logic.getCurrentController()
    scene = bge.logic.getCurrentScene()
    own = cont.owner
    keyboard = bge.logic.keyboard
    JUST_ACTIVATED = bge.logic.KX_INPUT_JUST_ACTIVATED
    JUST_ACTIVATED2 = bge.logic.KX_INPUT_ACTIVE

    MenuBack = scene.objects['MenuBack']
    MenuText = scene.objects['MenuText']
    MenuTextBack = scene.objects['MenuTextBack']
    Servo = cont.actuators["Servo"]


    own.localPosition.x = gl.PosiX
    own.localPosition.y = gl.PosiY
    own.localPosition.z = gl.PosiZ

    if keyboard.events[bge.events.DOWNARROWKEY] == JUST_ACTIVATED:
        if gl.globalDict["GameMenu"] == 3:
            gl.globalDict["GameMenu"] = 0
        gl.globalDict["GameMenu"] = gl.globalDict["GameMenu"] + 1
    if keyboard.events[bge.events.UPARROWKEY] == JUST_ACTIVATED:
        if gl.globalDict["GameMenu"] == 1:
            gl.globalDict["GameMenu"] = 4
        gl.globalDict["GameMenu"] = gl.globalDict["GameMenu"] - 1

    if gl.globalDict["GameMenu"] == 1:
        MenuTextBack.localPosition.y = 0.17
    if gl.globalDict["GameMenu"] == 2:
        MenuTextBack.localPosition.y = -0.08
    if gl.globalDict["GameMenu"] == 3:
        MenuTextBack.localPosition.y = -0.32

    if keyboard.events[bge.events.RETKEY] == JUST_ACTIVATED:
        if gl.globalDict["GameMenu"] == 1:
            gl.globalDict["Start"] = True
            gl.globalDict["GameMenuAct"] = False
            MenuBack.visible = False
            MenuText.visible = False
            MenuTextBack.visible = False
        if gl.globalDict["GameMenu"] == 3:
            gl.globalDict["Loading"] = False
            gl.globalDict["meta1"] = False
            gl.globalDict["meta2"] = False
            gl.globalDict["meta3"] = False
             gl.globalDict["meta4"] = False
             gl.globalDict["UltimoTiempo"] = '"00:00.00"'
             gl.globalDict["MejorTiempo"] = 0
             gl.globalDict["MejorTiempo2"] = 99999
             gl.globalDict["Incl.Sumatori"] = 0
             gl.globalDict["Incl.Max"] = 0.0000
             gl.globalDict["Incl.Max2"] = 0
             gl.globalDict["MenuPos"] = 1 #0 = JUGAR, 1 = EN XARXA, 2 = OPCIONS
             gl.globalDict["PreMenuPos"] = 1 # Menu, anterior a este.
             gl.globalDict["MenuLabel"] = 1
             gl.globalDict["SelectPos"] = 1
             gl.globalDict["PreSelectPos"] = 1
             gl.globalDict["SelectLabel"] = 0
             gl.globalDict["SelectEnd"] = 8
             gl.globalDict["Bubble"] = 1
             gl.globalDict["Start"] = False
             gl.globalDict["GameMenu"] = 1
             gl.globalDict["GameMenuAct"] = False
             gl.globalDict["MenuStart"] = False
             scene.replace("Intro")
```

```
120.
121. #Variables globales
122.
123. def main():
124.     if gl.globalDict["GameMenuAct"] == False:
125.         move()
126.         inter()
127.     else:
128.         PauseMenu()
129.         ActMov.dRot = [0.0,0.0,0]
130.         ActMov2.dRot = [0.0,0.0,0]
131.         ActMov3.dRot = [0.0,0.0,0]
126.         if own["ori"] < -0.05: #velocitat de gir maxima.
127.             own["ori"] = -0.05
128.
129.
130.         #IMPRESSIO
131.         if SenSorra.positive == True: #Deteccio de
    materials lents.
132.             MatLent = True
133.         if SenJespa.positive == True:
134.             MatLent = True
135.
136.         if MatLent == False:
137.             if SenAsfalt.positive == False:
138.                 Servo.linV = [0,own["acc"] /1.05 ,0]
139.                 ActMov.dRot = [0,0,0]
140.             if SenAsfalt.positive == True: #Deteccio de
    vol.
141.                 Servo.linV = [0,own["acc"],0]
142.                 ActMov.dRot = [0.0,0.0,own["ori"]]
143.
144.         if MatLent == True:
145.             Servo.forceLimitY = [ -20.0, 20.0, True]
146.             ActMov.dRot = [0.0,0.0,own["ori"]]
147.             own["acc"] = own["acc"] / 1.04
148.             Servo.linV = [0,own["acc"],0]
149.
150.         #So
151.         Sound1.volume = 0
152.         cont.activate(Sound1)
153.         if own["acc"] > 0.02:
154.            Sound1.volume = own["acc"] / 2 *
    gl.globalDict["VolumSo"]
155.            Sound1.startSound()
156.         if own["acc"] < 0.02:
157.            Sound1.stopSound()
158.
159.         #Metes
160.         if SenMeta.positive == True:
161.             gl.globalDict["meta1"] = True
162.         if SenMeta2.positive == True:
163.             gl.globalDict["meta2"] = True
164.         if SenMeta3.positive == True:
165.             gl.globalDict["meta3"] = True
166.         if SenMeta4.positive == True:
167.             gl.globalDict["meta4"] = True
168.
169.         #Rotacio endavant
170.         acc2 = own["acc"]
171.         acc2 = acc2/2
172.
173.         if SenAsfalt.positive == False:
174.             acc2 = acc2 * 4
187.                 max = +1
188.
189.         if suma >= 30: #Limits de velocitat
190.             suma = suma - 1
191.             max = 0
192.         if suma <= -30:
193.             suma = suma + 1
194.             max = 0
195.
196.         gl.globalDict["Incl.Sumatori"] = suma
197.
198.         #IMPRESSIO (Inclinacio)
199.         realpos = gl.globalDict["Incl.Max2"]
200.         realpos = realpos + max
201.
202.         gl.globalDict["Incl.Max"] = max / 100
203.         gl.globalDict["Incl.Max2"] = realpos
204.         ActMov3.dRot = [0,max/50,0]
205.
206.     Musica.volume = gl.globalDict["VolumMusica"]
207.
208. def inter():
209.
210.     JUST_ACTIVATED = bge.logic.KX_INPUT_JUST_ACTIVATED
211.     JUST_ACTIVATED2 = bge.logic.KX_INPUT_ACTIVE
212.
213.     MenuBack = scene.objects['MenuBack']
214.     MenuText = scene.objects['MenuText']
215.     MenuTextBack = scene.objects['MenuTextBack']
216.     Servo = cont.actuators["Servo"]
217.
218.     #Declaracion de variables y demas
219.     print1 = scene.objects["Timer1"]
220.     print2 = scene.objects["Velocity"]
221.     print3 = scene.objects["TimeOut1"]
222.     print4 = scene.objects["TimeOut2"]
223.
224.     #Menu
225.     if keyboard.events[bge.events.ESCKEY] ==
    JUST_ACTIVATED2:
226.         MenuBack.visible = True
227.         MenuText.visible = True
228.         MenuTextBack.visible = True
229.         #gl.globalDict["Start"] = False
230.         gl.globalDict["GameMenuAct"] = True
231.
232.         gl.PosiX = own.localPosition.x
233.         gl.PosiY = own.localPosition.y
234.         gl.PosiZ = own.localPosition.z
235.
236.     #Tiempo contrarreloj
```

Actualment (12/1/2013) Bubble Racer es troba en la seva versió 0.33, on el codi font ha estat àmpliament reformat, podeu trobar totes dues versions en el CD adjunt.