Zoper

Un joc que activarà la teva ment i el teus reflexos

Treball de final de carrera (TFC)

Nom: Ismael Gonzàlez Burgos

Index de contingut

1. Disseny del joc

La idea principal es fer un joc semblant al joc Zoop que va sortir a la seva època per a SEGA Megadrive, SuperNintendo (SNES) i PC. Després es va fer un remake a PlayStation i Sega Saturn.

La historia de aquest joc es bastant senzilla, es tracta d'evitar que els enemics o fitxes arribin fins al centre del tauler on es troba la nostra fitxa, que es el que controlarem al joc. Si arriben, el joc s'acaba. Tindrem també diferents modes de joc per a que les partides no siguin sempre iguals. Ni que el joc es faci massa repetitiu.

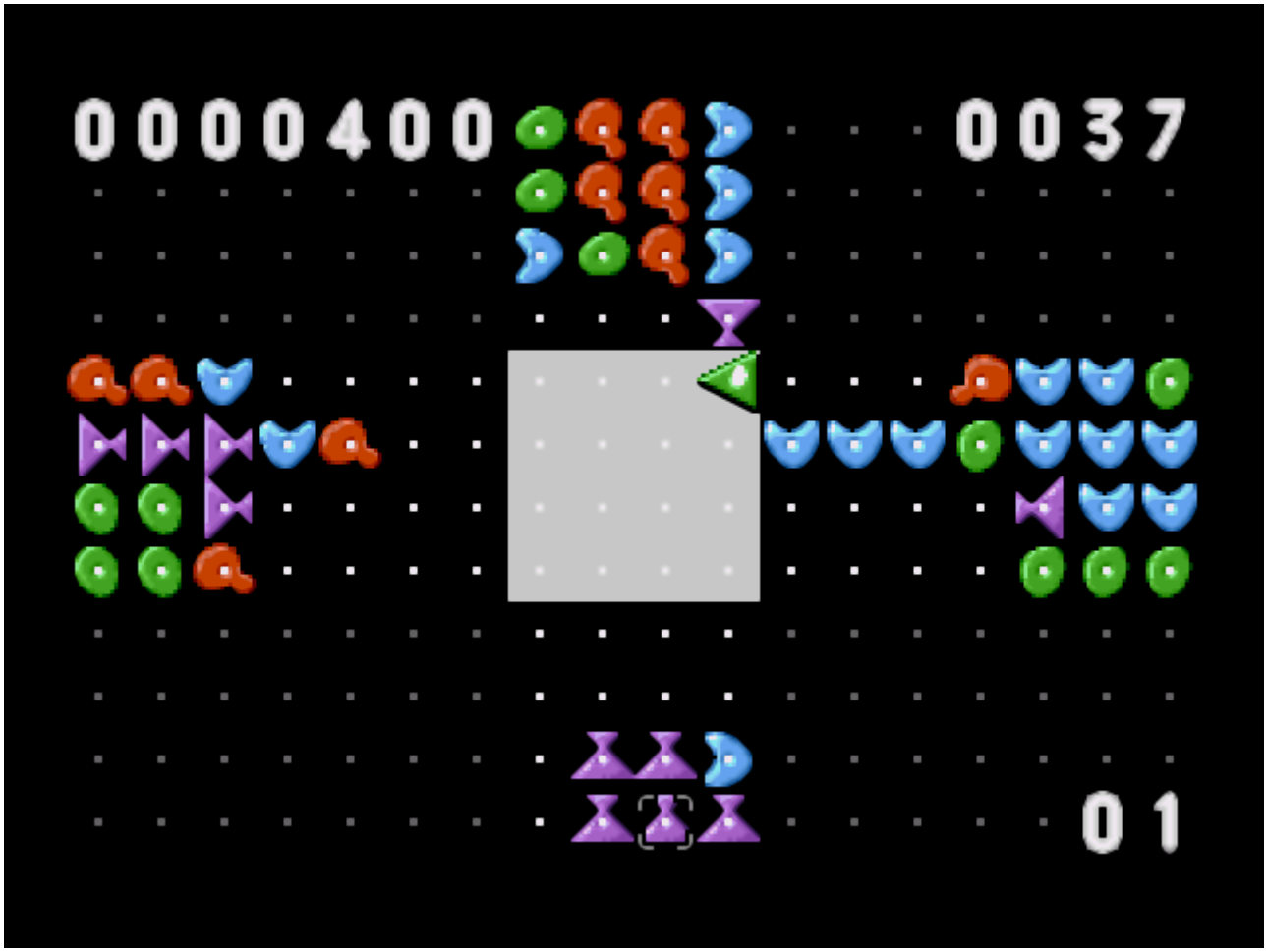


Figura 1: El joc Zoop per a MegaDrive

1.1 Història

La historia del joc és ben senzilla, ja que només hem d'evitar que les fitxes arribin al centre o es

troba el jugador. Aquesta senzillesa ha fet que aquest joc encara sigui recordat i jugat per les noves generacions. Potser podria haver estat més famós que el tetris (opinió personal) si hagués tingut la publicitat adequada i una mica més de sort.

La pantalla principal del joc incorpora un taulell amb un taulell mes petit a l'interior. Aquest taulell intern es per on es podrà moure el jugador.

1.2 Regles del joc

Explicarem aquí, de manera general, com funciona el joc i quines son les seves regles.

Comencem amb el tauler buit a excepció de e es troba al centre del taulell. La fitxa es pot moure per aquest quadrat central amb llibertat.

Les fitxes que no són les del jugador que van apareixent als voltants del quadrat central són generades de manera aleatòria. La posició on apareixerà serà el més allunyat possible del centre a una fila o columna al atzar. El tipus de fitxa, d'entre totes les disponibles també es generada de manera aleatòria.

La nova fitxa que aparegui farà moure totes les altres de la seva fila o columna cap al centre en un a posició. Com ja s'ha mencionat, si una de les fitxes arriba al quadrat central, la partida acaba. Per evitar que les fitxes arribin al centre el jugador pot disparar la seva fitxa cap a una fila o columna del taulell. Això farà, depenent del seu color i les de les fitxes que hi hagi, que certes fitxes se'n vagin.

Podem veure doncs, que l'objectiu del joc és aguantar el màxim de temps possible, esborrant les fitxers del taulell el més ràpid possible.

1.2.1 Moviment del jugador pel taulell

El jugador controla la fitxa que es mou al centre quadrat central del taulell. No pot sortir d'aquest quadrat.

La fitxa del jugador es controlada mitjançant cinc tecles. Quatre son tecles de direcció i una es la

tecla per a disparar la fitxa contra els enemics.

1.2.2 Moviment de la resta de fitxes pel taulell

La resta de fitxes estan situades al voltant del quadrat central a on es mou el jugador. Aquestes fitxes estan agrupades a algun dels costats del quadrat del jugador I van apareixent una darrera l'altre en direcció cap al centre. Les fitxes formen part d'algun dels grups que es situen rodejant el quadrat central. Aquestes fitxes apareixen a un grup, a una línia i a una columna, i empenyen la resta de fitxes de la seva fila o columna cap al centre. Quan una d'aquestes fitxes es fica en la zona central del taulell, el joc acaba.

1.2.3 Eliminant fitxes del taulell

Per poder avançar en el joc, hem d'anar eliminant fitxes el mes ràpidament que podem a mesura que vagin apareixent.

Per a poder eliminar les fitxes hem d'impactar amb la fitxa del jugador amb una (o un grup) que tinguin el mateix color.

Les fitxes s'esborraran del tauler seguint la lògica següent:

Es comprovarà la primera fitxa que es trobi la fitxa del jugador. Si son del mateix tipus (color i forma) passarà a ser una casella buida i comprovarà la següent en direcció contraria al centre del taulell.

Quan trobi una fitxa que no és del tipus de la fitxa del jugador, els tipus de les dues fitxes seran canviats, es a dir, la fitxa del jugador passarà a ser el de la fitxa trobada i la fitxa trobada agafarà el tipus de la fitxa del jugador.

Si la fitxa del jugador arriba al final per un costat, a dalt o a baix tornarà al tauler central sense canviar de tipus i havent esborrat totes les fitxes de la fila o columna.

Quantes més fitxes s'esborrin d'una tacada (amb un sol moviment/llançament) més punts pujaran al marcador.

1.3 Modes de joc i dificultat

Els modes de joc a implementar són encara un tema a pensar. Com a mínim m'agradaria implementar els del joc original, però evidentment tinc més idees per a modes de joc. Dependré una mica del temps disponible i de si és possible implementar-los o no. Més informació en les següents entregues.

Tornant als modes bàsics del joc:

- Mode token: En aquest mode haurem d'eliminar fitxes enemigues per passar de nivell.
- Mode «time»: Aquí també tindrem nivells, però al passar d'un a l'altre haurem de resistir un cert temps.

També podrem establir al començament del joc la dificultat/velocitat inicial.

1.4 Personatges

Com es evident, en aquest joc no tindrem cap personatge humà o humanoide. Aquí les protagonistes son les fitxes o, com es deien al joc original «zoops». Amb únicament dos tipus bàsics, el jugador i els enemics que intenten arribar al centre.

1.5 Gràfics i sons

Donat que es tracta d'un joc senzill no es necessitarà una gran quantitat de recursos multimèdia. De fet, intentaré fer-los jo tot sol.

El gràfics que pugin ser necessaris es generaran mitjançant codi o bé s'utilitzarà alguna web de recursos lliures. Respecte els sons i la música, estic pensant entre utilitzar els recursos lliures també o extreure els sons del joc original. Al final no he afegit sons al projecte, però podria ser una bona ampliació.

Tot aquest apartat 1ha estat elaborat abans de començar el projecte, però modificat a posteriori.

2. Disseny i codificació

2.1 Conceptes bàsics del disseny del codi

La idea bàsica en el disseny i la implementació del codi del joc és la seva orientació a objectes.

He escollit el llenguatge de programació C++ per a la implementació per diferents motius:

- És el llenguatge de programació en el que personalment, tinc més experiència professional i personal.
- És orientat a objectes (no completament com el Java o el C#) però si que permet fer un projecte completament orientat a objectes, i més amb les noves característiques de C++11[1].
- Per a programar jocs (especialment d'escriptori, encara que també per mòbils) és el llenguatge que ofereix més eficiència, evidentment a canvi d'un control explícit de la gestió de recursos però molt més efectiu de cara a aplicacions que normalment requereixen un gran consum de CPU i GPU.

El projecte s'ha desenvolupat utilitzant Visual Studio 2013 i 2015 en alguns moments. També l'extensió Vissual Assist X ha estat utilitzada.

2.2 Estructura de directoris del projecte

El projecte està estructurat en una solució de Visual Studio amb un projecte associat a ella. La llibreria pròpia que ha estat desenvolupada per a ser utilitzada al joc es troba dins del directori lib. Aquesta llibreria es troba dividida en directoris que contenen els espais de noms (nanemspaces) i les classes.

2.3 Patrons i metodologies de disseny

El projecte en general ha estat dissenyat utilitzant orientació a objectes. Això ens permet utilitzar una sèrie de patrons de disseny ja establerts i coneguts en el disseny de certes funcionalitats del joc. Intentaré enumerar els més importants i posar alguns exemples amb codi del projecte. Aquesta secció no pretén donar la definició dels patrons en si mateixos, per a consultar-la, podeu veure la

bibliografia ().

2.3.1 Factory

Aquest patró ens permet crear objectes des d'un objecte d'un tipus específic des d'un objecte pare o «Factory». En fem ús a l'hora de crear nous nodes gràfics per part del motor. El pseudocode utilitzat és:

```
NodeGrafic NodeGràfic::creaNodeGràfic(string nom);
```

aquesta construcció és típic del ptrò «Factory». El fet de que un node gràfic crei als seus fills és típics dels motors gràfics que treballen amb *trees* (arbres) com el que hem construït a la llibreri interna.

2.3.2 Adapter

Aquest patró també es conegut per *wrapper* (envoltori) es utilitzat normalment per adaptar (d'aquí el seu nom) la interfície d'una classe a alguna classe comuna per a que un altre sistema o classe que només utilitza aquesta interfície la pugui utilitzar.

Per exemple, al nostre projecte hem adaptat algunes classes de SFML a la interfície comuna del nostre motor.

2.3.3 Facade

Aquest patró de disseny ens permet amagar la complexitat d'alguns components mostrant a l'usuari una interfície comuna. La he utilitzat per exemple per amagar sf::Text i RenderNode «dins» de Renderizable.

2.3.4 Composite

Aquest patró és el que ens permet especificar per exemple el mètode draw de IDrawable a cada classe específica per a què realitzi el dibuix del component específic que defineix.

2.3.5 RAII

Aquesta metodologia es molt útil sobretot en llenguatges on els recursos es reserven i alliberen explícitament com és el C++.

RAAI (Resource Acquisition Is Initialization)[2] és un mètode de desenvolupament que ens permet desenvolupar programes d'una manera més segura per evitar els *memory leaks* o forats de memòria englobant l'adquisició de recursos als constructors i l'alliberament al destructors.

Hem de notar que quan dic recursos no em refereixo només a fitxers de dades, imatges o altres recursos grans, reservar 4 bytes per un int també és una adquisició de recursos i també ens pot portar a un memory leak. Les variables o propietats d'una classe que no utilitzen el heap (que no son punters), s'allotjaran a la pila i el mateix llenguatge gestionarà el RAII (sense que fem res en cas dels tipus simples definits pel llenguatge, o bé via cridant el destructor en cas de tipus definits per l'usuari).

L'antic problema del C++ per fer RAII complet era l'absència a l'estàndard de *smart pointers*. Amb el nou estàndard C++11 podem triar entre diferents tipus de «apuntadors intel·ligents». Al projecte utilitzo bàsicament dos (tres si contem el weak_ptr com a un altre tipus) tipus d'apuntadors:

- (std::)unique_ptr<T> : Es un tipus que encapsula un punter únic. És a dir, només es podrà fer ús del punter dins del seu *scope* i serà destruït automàticament quan arribem al final de l'scope.
- (std::)shared_ptr<T>: Aquest tipus també encapsula un apuntador que pot ser compartit per diferents classes. El tipus incorpora un comptador de referencies, de manera que el apuntador serà destruït automàticament només quan no quedi cap apuntador apuntant a la dada. Això ens porta quasi al funcionament d'un garbage collector, però encara tenim una altre avantatge: podem crear weak_ptr<T> des d'un shared_ptr<T>.
- (std::)weak_ptr<T>: Aquest tipus es crea a partir d'un shared_ptr<T> i és un apuntador a les dades de shared_ptr<T> però no compta en el comptador de referències de shared_ptr<T>. Això fa que per utilitzar-lo, primer haurem d'obtenir un shared_ptr<T>. És útil per guardar apuntadors que volem observar, però no volem controlar la seva durada.

3. Ús de la llibreria SFML

La llibreria SFML (Simple and Fast Multimedia Library)[3] ha estat utilitzada per desenvolupar la part multimèdia del projecte. La llibreria es divideix en mòduls on cadascú s'encarrega de la gestió de determinades classes relacionades. Per exemple, trobem el mòdul de gràfics o de xarxa.

No tots els mòduls de la llibreria han estat necessaris per al desenvolupament del projecte. Cada mòdul proveeix d'un fitxer de llibreria d'enllaç dinàmic (.dll en Windows, .so en GNU/Linux) amb dues versions: acaba en «-d» que significa versió de *debug* o bé sense aquesta terminació on indica que es tracta de la versió de *release*.

Encara que la llibreria pot ésser enllaçada de manera estàtica, he utilitzat la opció d'utilitzar-la amb enllaç dinàmic per evitar forçar la instal·lació de totes les dependències necessàries per realitzar la compilació, i per a donar la opció de substituir la llibreria per una opció més nova. (Això funciona si les versions són compatibles a nivell binari, però és una de les avantatges de l'enllaç dinàmic de llibreries).

Encara que per mitjà de la llibreria interna desenvolupada he intentat evitar l'ús directe de sfml en el codi del joc, en alguns casos ha estat impossible evitar aquest ús. Totes les referències a classes de sfml van precedides del namespace sf::, no hem utilitzat el «using namespace sf» perquè sigui fàcil trobar i seguir aquestes referències i en una futura iteració del projecte substituir-les per codi de la llibreria interna.

3.1 Principals usos de la llibreria SFML al projecte

La llibreria SFML ens prové de diferents entitats per a *renderitzar* objectes a la pantalla. Aquesta renderització es realitza utilitzant OpenGL internament.

La llibreria interna crea un wrapping (RenderNode) amb una unió de les dues classes que utilitzem per a dibuixar. Les classes de dibuix de SFML implementen la interfície sf::Drawable, que indica que una classe pot ser dibuixada en un sf::RenderTarget. La majoria de les classes per a dibuixar també tenen herència de sf::Drawable per a poder ser dibuixades a un sf::RenderTarget i de sf::Transformable per poder rebre transformacions bidimensionals (escala, posició, rotació) per a decidir el resultat final de la renderització.

4. Codi del projecte

4.1 La llibreria interna

4.1.1 Utilitats

Pel desenvolupament d'aquest joc, encara que per dibuixar sprites i nodes a pantalla hem utilitzat la llibreria SFML com ja s'ha esmentat, per a moltes tasques comunes a la programació de jocs que he usat i implementat, he creat una llibreria interna, que es troba completament encapsulada al namespace lib, i també separada al directori lib. Aquesta llibreria conté uns quants components, alguns de molt comuns a la programació de jocs i algunes utilitats. Destacaré algunes de les classes i mòduls.

- Sistema de tipus simples propis: La llibreria interna incorpora redefinicions del tipus simples. Això és útil si es pretén que la llibreria sigui algun dia multiplataforma. Això es pot portar molt més enllà i incorporar una petita llibreria que substitueixi la STL per no haver de dependre d'ella com fa, per exemple Irrlicht.

- Sistema de gestió del taulell: Sota el namespace lib::board es troba un component per a la gestió de un taulell de (m x n) caselles. Ens proporciona una API per crear elements al taulell, moure'ls, esborrar-los i poder rebre callbacks en tots els cassos mencionants.

- Generació de numero pseudoaleatoris: Classe per ajudar a fer aquesta generació, encapsulant la inicialització i gestió del generador pseudoaleatori.
- Logs: Sistema de logs integrat i que desactivat per compilació. Utilitza multithreading amb threads natius de C++11 sense llibreries externes. També permet gravar els logs a un fitxer.
- Pclock: Rellotge igual que el que proveeix SFML però amb un mètode per poder fer pausa.
- Gestió de finestres: Encapsulant classes de SFML ofereix facilitats per crear i dibuixar automàticament en finestres, així com un comptador de frames per segon.
- Controlador de programes: Encapsula la classe principal controladora de tot el joc dins d'una classe, el que ens permet de tenir un punt de començament comú i una lògica de programa comuna.
- Animacions: Mitjançant timers proveeix d'un senzill però efectiu sistema d'animacions per posicions i colors. El sistema es base en que donat un valor de començament, de final i un temps, es pot generar un delta que es pot utilitzar per a calcular el valor correcte en un temps donat. Està realitzat mitjançant templates.

4.1.2 Sistema de renderització

Encara que la llibreria SFML ja prové un sistema per a dibuixar diferents nodes com sprites, shapes i textos, la llibreria interna prové d'una façana integrant els diferents tipus de nodes utilitzats dins d'un mateix sistema de renderització i, el que és més important amb unes propietats comunes.

Això ha estat necessari per una raó principal: el sistema d'herència utilitzat per SFML (que personalment, trobo que és un problema). Ho explicaré amb un exemple:

Les clases sf::CircleShape i sf::Text, tenen herència de sf::Drawable i sf::Transformable, les dues. Primer de tot, tenim el problema que no hi ha una classe comuna, una cosa com sf::DrawableTransformable o una cosa semblant. El problema del patró del rombe no apareixeria, perquè no tenen mètodes comuns, un s'encarrega de proveir mètodes per dibuixar i un altre de proveir mètodes per transformar-se a l'espai de coordenades. Aquest problema, fa que a l'hora d'estendre, si estenem sf::Drawable o sf::Transformable, l'objecte no serà d'algun dels tipus. Si decidim estendre una classe concreta com sf::CircleShape, perdrem la possibilitat de la generalització. Donat que tots els objectes dibuixables són sf::Drawable i sf:Transformable, hauria d'haver-hi una interfície comuna.

Per a solucionar aquest problema he creat una classe comuna com una façada i un union que conté un apuntador a una classe específica. Com la classe també és sf::Drawable i sf::Transformable, també es poden obtenir a partir d'aquesta. També implementa o modifica la interfície de sf::Drawable on ha estat necessari.

4.1.3 Gestió d'escenes

Per a la gestió de la lògica de decisió de nodes a dibuixar, el petit motor gràfic que incorpora la llibreria interna implementa un petit sistema de gestió d'escenes en arbre. La gestió d'escenes en arbre és típica dels motors gràfics en 3D on les matrius de cada node es multipliquen per les dels seus fills. D'aquesta manera, cada node té una matriu local i una matriu global. La local és la seva pròpia i la global és la global del seu pare multiplicada per la seva local.

En el petit motor d'escenes implementat en la llibreria local no han estat implementades totes les funcionalitats que acostumen a tenir aquests tipus de motors, però sí les més bàsiques i les necessàries per al joc.

- Transformacions en arbre: Com hem dit, la transformació d'un node depèn de la del seu pare.
- Cada node pot crear més nodes: que seran els seus fills.
- Renderització de tots els nodes a la vegada: Això depèn de la implementació de l'API de baix nivell de renderització accelera la renderització a pantalla.
- Selecció de l'escena actual: Podem tenir moltes escenes a memòria, però només una activa. Quan canviem l'escena activa, s'executa el mètode Scene::onExitScene() de la escena prèviament activa i el mètode Scene::onEnterScene() de la nova escena. Això ens facilita el control dels punts d'entrada i sortida de cada escena i poder establir invariants i controls.

4.1.4 Sistema de menús

Sistema de menús: Components per a la creació i gestió per a un menú per al joc. Incorpora pocs components, però són els que he necessitat per a aquest joc concret. Si la llibreria es convertís en un projecte separat seria un bon candidat per a ser ampliat. El programador pot crear diverses pantalles amb components i establir un camí entre elles depenent de les entrades que realitzi l'usuari.

El sistema de menús implementat es força senzill i inclús es pot veure en si mateix com una petita variació del sistema d'escenes o tenim un MenuStep que està actiu en cada moment.

4.1.5 Sistema de configuració

La llibreria interna també incorpora un sistema de gestió de la configuració que té tres utilitats essencials:

- Proveir d'una configuració de variables que poden ser llegides d'un fitxer i canviades directament des del fitxer (no recomanat) o mitjançant els menús del programa.
- Proveir de base per a altres components que necessitin llegir dades ordenades de l'estil clau i valor d'un fitxer. Exemple: el gestor de recursos.
- Servir com a diccionari per a passar dades clau/valor d'una entitat a un altre, que no es poden comunicar normalment mitjançant una interfície del programa. Com poden ser dos escenes o dos MenuStep.

La implementació de la configuració llegeix aquestes dades clau/valor d'un fitxer o be pot crear

un diccionari nou (per passar dades entre entitats). També és crea un diccionari nou si no es troba el fitxer. És decisió de l'usuari de la classe que fer si el fitxer no existeix, però l'API ens permet crear valors per defecte per a aquest cas.

La configuració guarda un diccionari estàtic amb totes les dades i quan es crea una instància amb un fitxer un id determinat busca si ja existeix. Si ja existeix, li dona accés a les mateixes dades i si no, crea un nou diccionari. D'aquesta manera, totes les classes que, per exemple, accedeixen al fitxer «config.cfg» ho fan a les mateixes dades i les modifiquen, no a una còpia.

4.1.6 Gestor de recursos

El gestor de recursos està construït per a poder llegir fitxers gràfics que utilitzi el joc. Aquests recursos es defineixen en un fitxer de configuració que llegim mitjançant herència del component Configuració. Com és natural, els fitxers han d'existir per a que funcioni el joc. Un cop els fitxers són carregats a memòria, ja poden ser utilitzats com a textures o fonts al joc.

La classe Resource també fa de façana amagant les diferencies entre una textura i una font, proveint d'una interfície comuna per al seu ús.

4.2 El codi del joc

El codi del joc utilitza, com és evident la llibreria interna que ha estat creada des de zero per a aquest joc (encara que ha estat desenvolupada com un petit projecte amb el que es podrien desenvolupar altres jocs).

Tenim diferents components principals i característiques:

- El joc està basat en escenes, que s'aniran canviant per ser actives. En aquest cas tindrem el menú i el joc en si mateix. Encara que aquestes escenes també tenen estats que són com escenes menys elaborades.
- El sistema és orientat a events. És a dir, s'intenta evitar fer pooling a l'entrada/sortida respecte el jugador. La llibreria SFML ja és orientada a events i això s'ha utilitzat per ha construir-hi a sobre.

4.2.1 Algorisme principal del joc

Intentaré explicar l'algorisme principal en que es basa el joc. No les regles, que s'expliquen en un altre apartat, si no l'algorisme que utilitza el programa per anar avançant.

4.2.2 Generació de noves fitxes

Quan arriba moment de genera una nova fitxa, llegim la quin dels quatre camins (esquerra, adalt, dreta, abaix). Quan sabem la zona generem un número a l'atzar per la coordenada que ens falta (a esquerra i dreta ens falta la línia i per dalt i baix ens falta la columna).

Un cop tenim la posició on sortida la nova fitxa, generem un altre número a l'atzar per saber el color (tipus) d'aquesta fitxa.

Quan tenim totes les dades que necessitem hem de moure totes les fitxes de la fila o columna cap al centre. Per això calculem la posició més propera al centre i comencem a moure'ns en direcció contraria cap a la posició de la nova fitxa.

A cada posició que trobem, si no és buida, mourem la fitxa cap al centre. Si la fitxa es troba ara, efectivament al centre, el joc a acabat. Si cap fitxa arriba al centre, al final haurem deixat la posició on volem col·locar la nova fitxa buida. Doncs bé, no tenim més que col·locar-la i esperar el moment de posicionar la següent fitxa.

4.2.3 Eliminació de fitxes

L'altre acció important del joc és la que realitza el jugador: l'eliminació de les fitxes per evitar que aquestes arribin al centre.

Quan el jugador prem la tecla de llançament, la fitxa central es troba en una posició i una direcció. Aquesta direcció serà utilitzada per arriba a la primera casella fora del quadre central.

A partir d'aquest moment, s'analitzarà cada casella fins trobar la primera que no sigui buida. Si la casella conté una fitxa del mateix color que la fitxa del jugador, la fitxa desapareixerà i se sumaran punts al jugador. Cada nova fitxa seguida del mateix color sumara més punts al jugador amb un multiplicador, incrementant el número de punts quantes més fitxes del mateix color seguides es

trobin.

Si es troba una fitxa de diferent color, aquesta canvia al color del jugador i el jugador canvia al color d'aquesta fitxa, és a dir, s'intercanvien els colors i la cerca de fitxes acaba.

Si s'arriba al final de la fila o columna sense haver trobat cap d'un altre color, el jugador manté el seu.

4.3 Estructura del projecte

Enumero aquí els principals directoris que es poden trobar al directori base de l'entrega

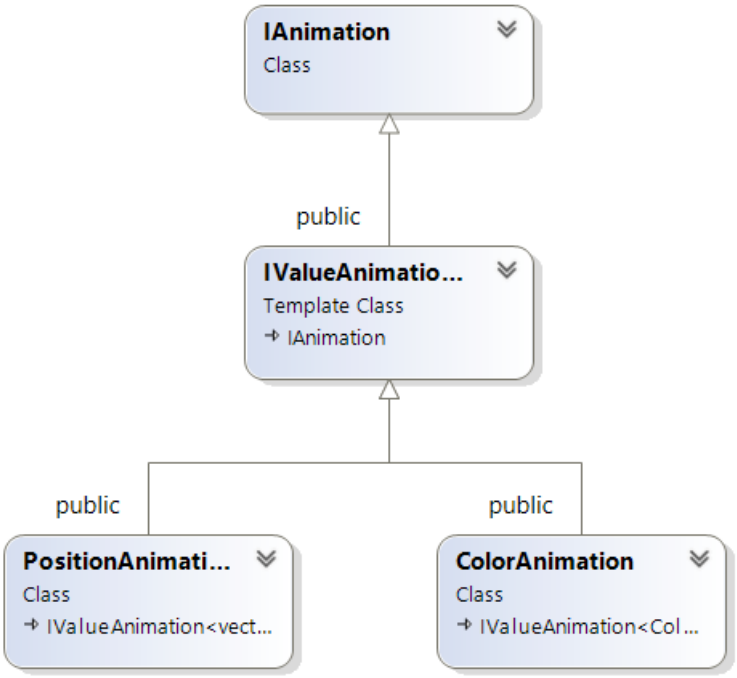
- Debug: Fitxers ja compilats en Debug
- Release: Fitxers ja compilats en Release
- Zoper: Projecte de Visual Studio 2013 que conté tot el codi del projecte
- Zoper/lib: Codi de la llibreria interna creada per a aquest projecte
- doc: Documentació del projecte
- install: Instal·lador del joc per Windows
- resources: Copia original dels recursos que utilitza el joc
- SFML: Fitxers d'include i llibreries SFML necessàries per a compilar el projecte.
- Dist: Directori de distribució del qual s'ha creat l'instal·lador i que pot servir per executar el joc sense instal·lar-lo.

4.3.1 Instal·lador

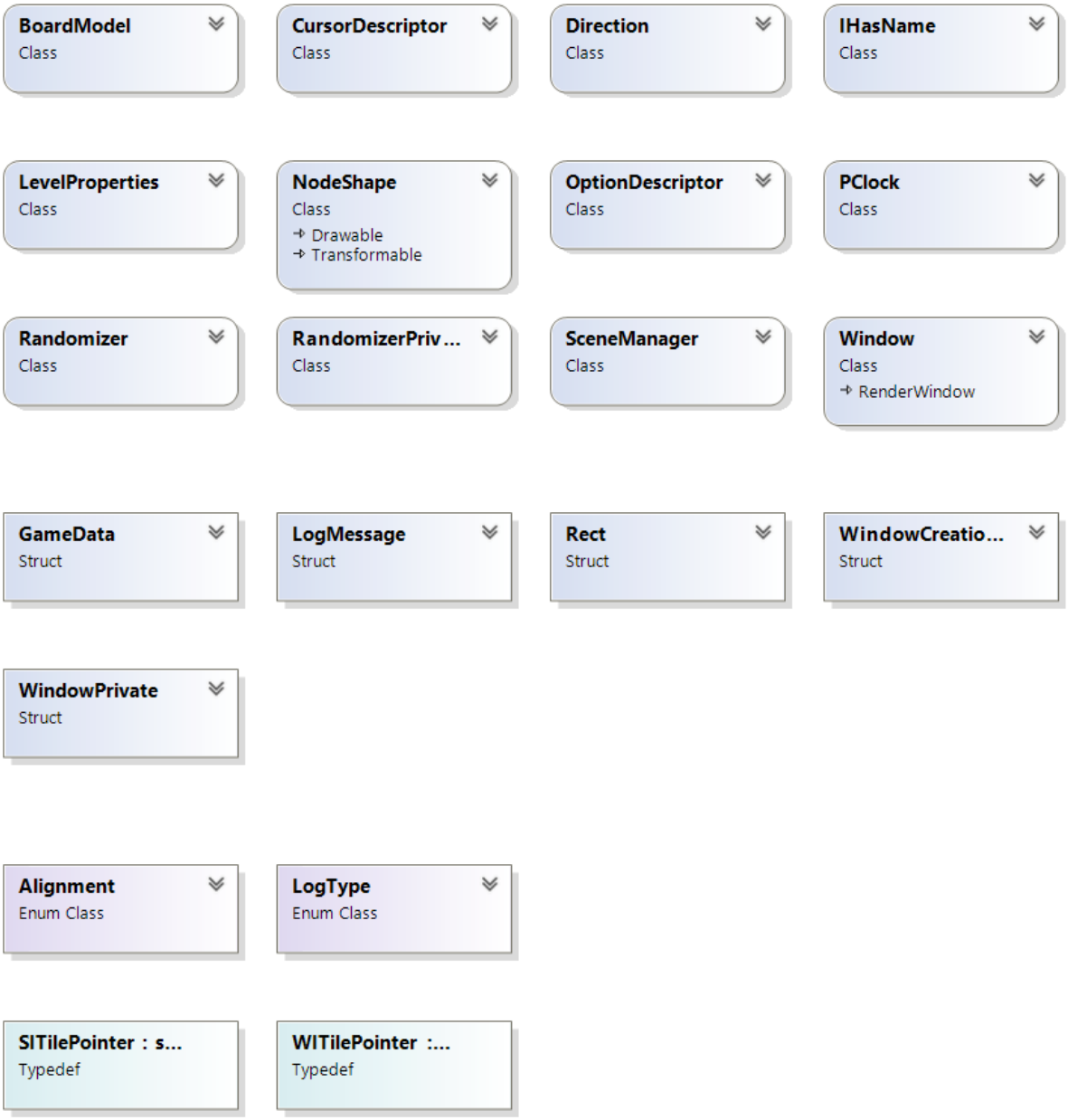
Mitjançant installForge he creat un instal·lador del programa. Instal·la la versió de debug (amb logs) i Release, per que es pugin provar les dues sense compilar.

4.4 Diagrama de classes

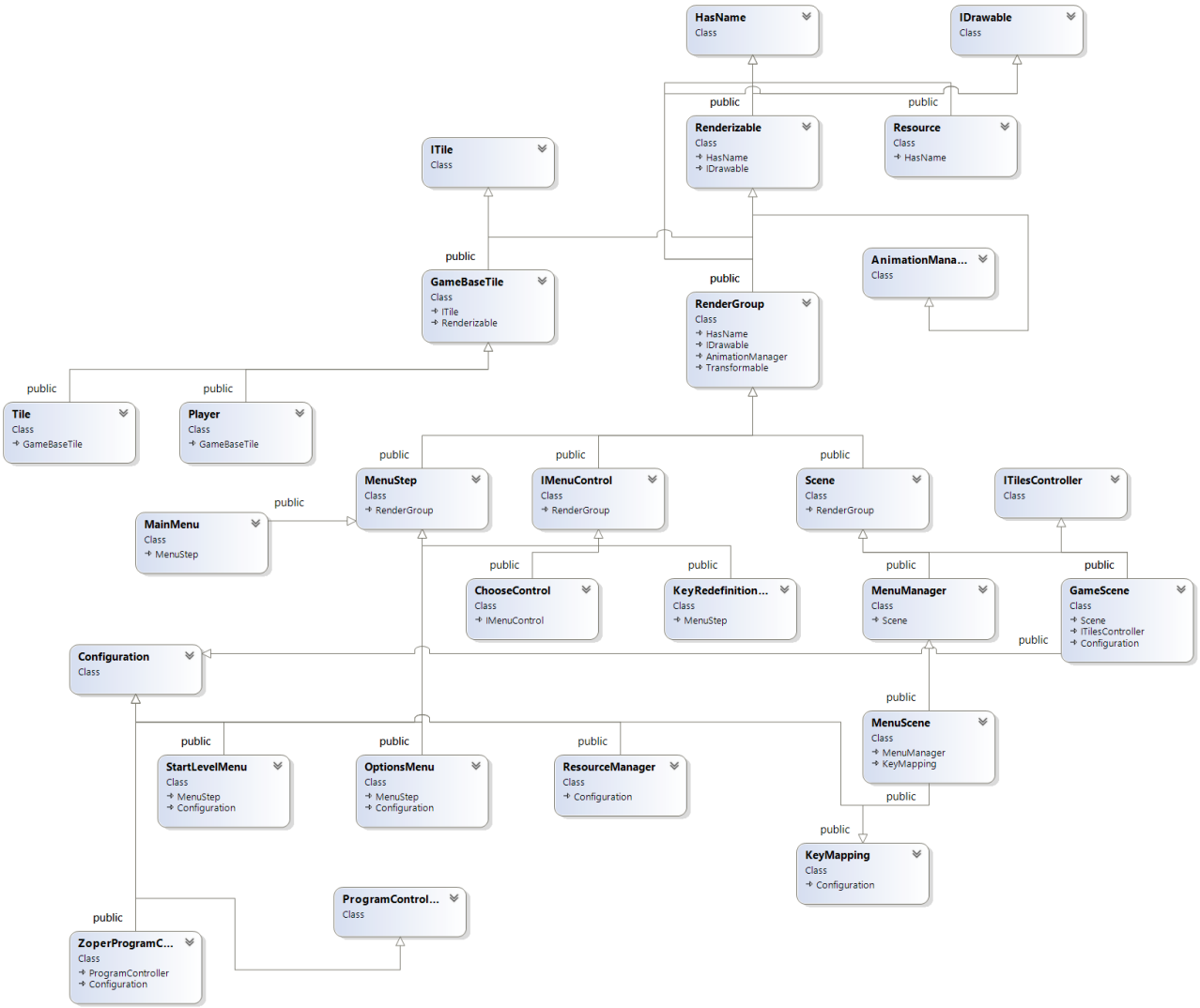
Podem veure aquí el diagrama de classes de l'aplicació:



Il·lustració 1: Part 1 del diagrama de classes



Il·lustració 2: Part 2 del diagrama de classes



Il·lustració 3: Part 3 del diagrama de classes

5. Pantalles i descripció del joc

5.1 Menú principal

El joc comença a la pantalla principal, que és un menú on podrem escollir entre en quin dels modes de joc voldrem participar, anar a les opcions o sortir:

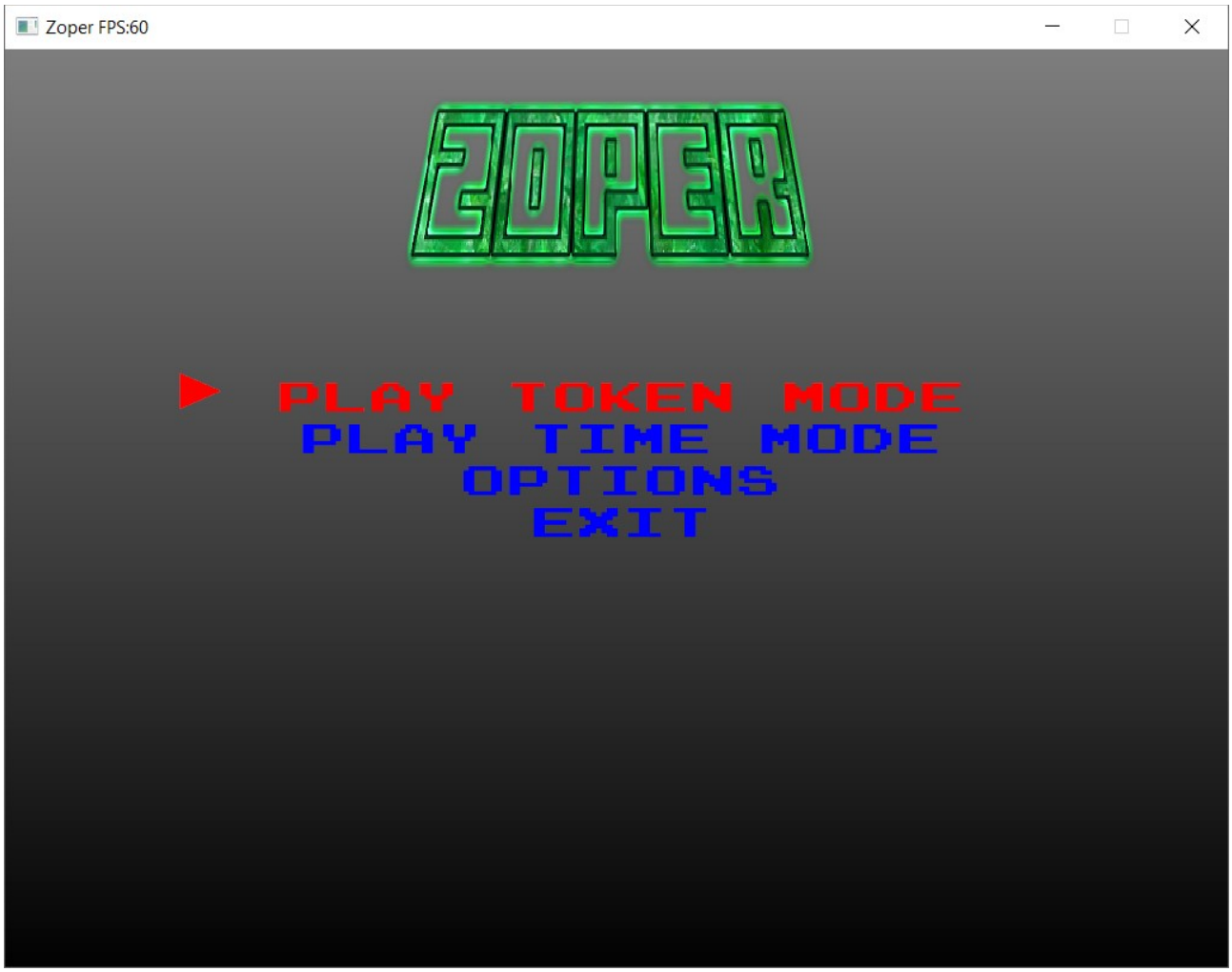


Figura 2: Menú principal del joc

En aquest i els altres submenús del joc les tecles que valen són les fletxes i el <ENTER> i no les utilitzades pel joc. Això és així per a facilitar que en cas que la redefinició del teclat sigui equivocada ens sigui molt difícil tornar a redefinir-lo. Aquest sistema s'utilitza en molts jocs, i en alguns que no, es complica molt arribar al menú en cas de redefinició incorrecte.

5.2 Menú d'opcions

Si seleccionem el menú d'opcions, anirem la pantalla d'opcions:

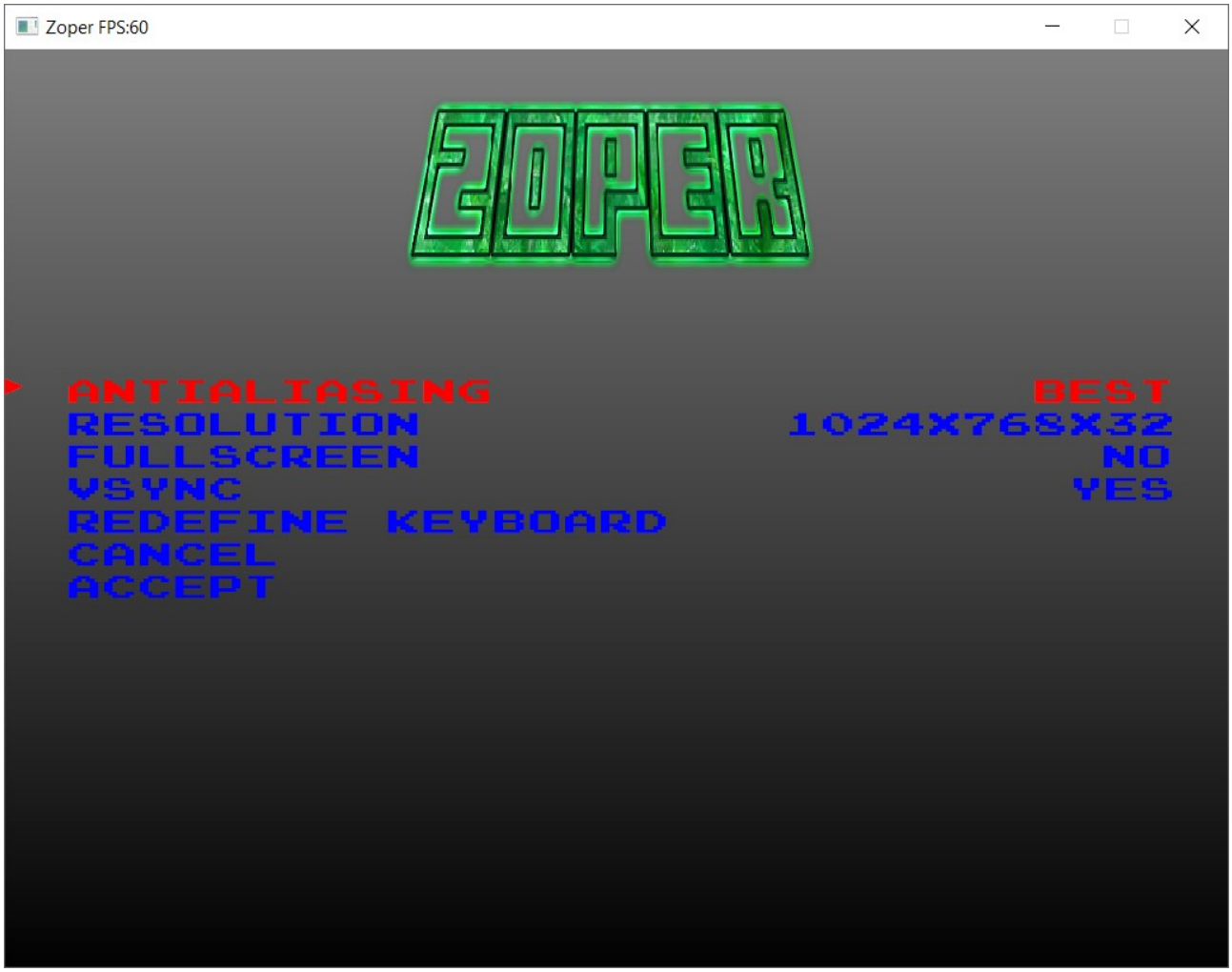


Figura 3: Menú d'opcions

Aquest menú ens permet canviar diferents opcions del programa:

- Antialiasing: Nivell de qualitat gràfica del joc. Depenent de la nostra tarja gràfica pot ser que no totes les opcions siguin suportades, és a dir, que no notem diferències.
- Resolution: Ens permet canviar la resolució de pantalla que serà utilitzada per mostrar el joc.
- Fullscreen: Aquí podem decidir si volem jugar a pantalla completa o no.
- Vsync: Indica si per redibuixar la pantalla esperarem a que el hardware ens avisi de que el flux d'electrons està lliure per a fer un update de la pantalla. Aquesta opció realment només funciona a pantalla completa, però inclús en finestra limitarà els frames per segon a 60 com a màxim.
- Redefine keyboard: Ens porta a la opció de redefinir el teclat.

- Cancel: Anul·la tots els canvis fets al menú i torna al menú principal.
- Accept: Accepta els canvis fets al menú (els grava mitjançant la classe lib::Config a config.cfg)

5.3 Redefinició del teclat

Si al menú d'opcions anem a «Redefine Keyboard» arribarem al següent menú:

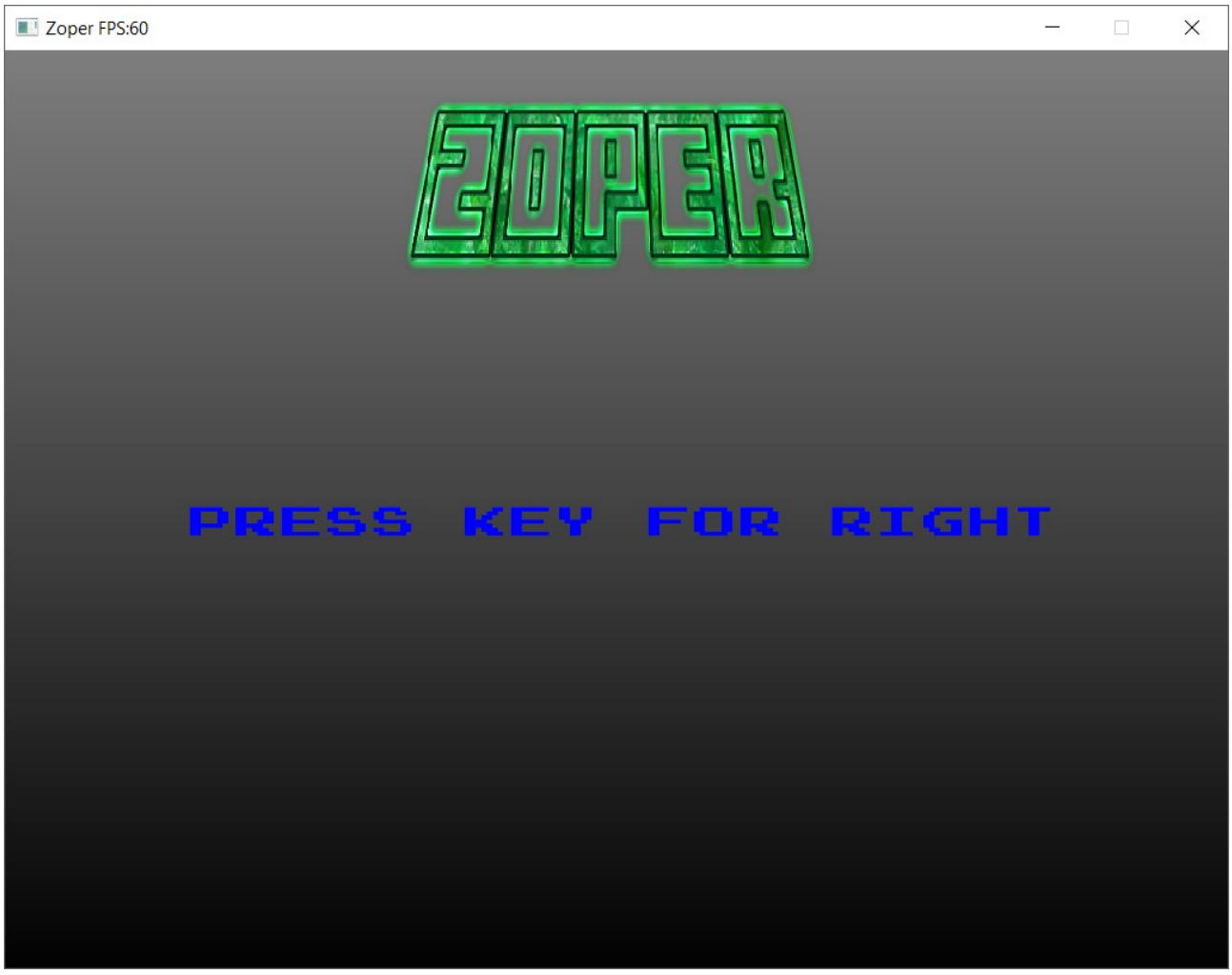


Figura 4: Menú per redefinir el teclat

Aquest menú és simple i ens va demanant que premem les tecles per a cada acció del joc. Evita que entrem duplicitats. Després tornarem al menú d'opcions. Només es gravarà si al menú d'opcions seleccionem «Accept».

5.4 Menú de selecció de nivell

Un cop seleccionem el mode en que volem jugar, podrem escollir a quin nivell començar:

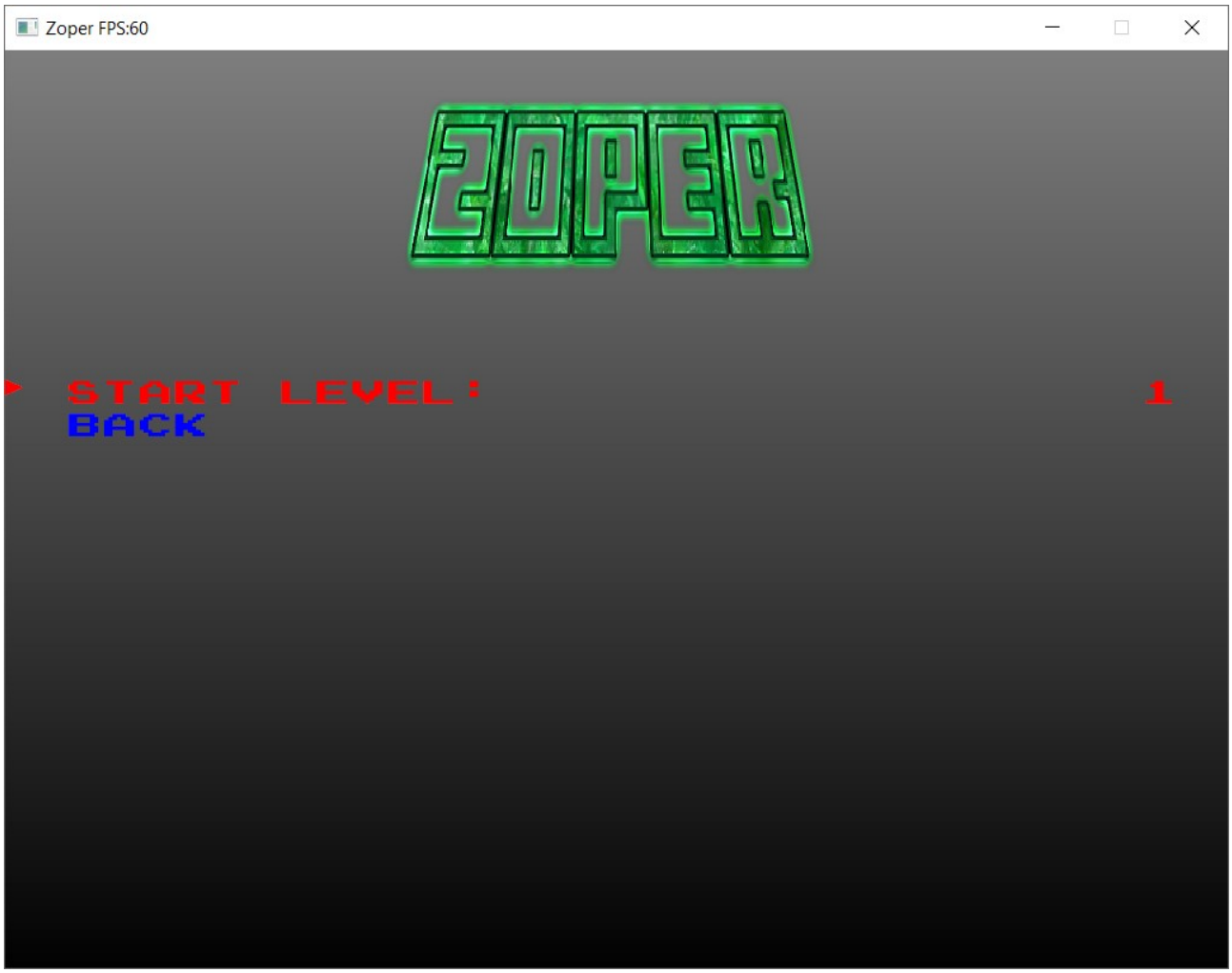


Figura 5: Menú de selecció del nivell on començar

Aquest menú apareix quan seleccionem en quin mode volem jugar i ens permet escollir un nivell (una dificultat) a la qual començar a jugar. Podem tornar endarrere o començar el joc.

5.5 Jugant al mode «Token»

El primer dels modes de joc és el mode «Token» on, per avançar de nivell haurem d'eliminar un número de fitxes. Cada cop que complim aquest objectiu avançarem al següent nivell i un nou objectiu de número de «Tokens» a destruir apareixerà. L'objectiu de «Tokens» a destruir s'anirà

incrementant a mesura que avancem nivells, l'objectiu de «Tokens» a destruir anirà augmentat. Així com la velocitat a la que surten els «Tokens».

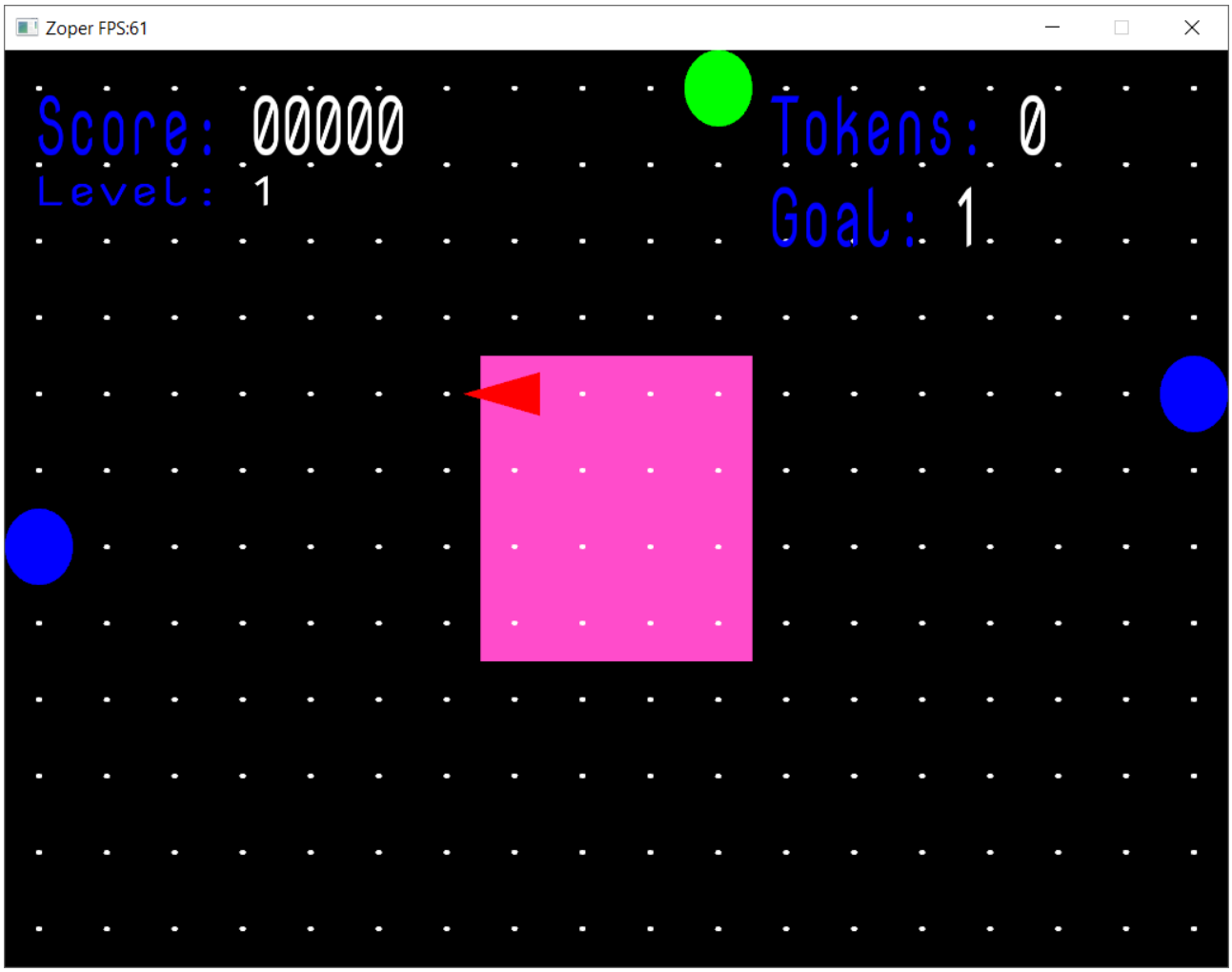


Figura 6: Jugant al mode "Token"

5.6 Jugant al mode «Temps»

En aquest mode de joc, en lloc d'intentar arribar a un número de fitxes per a completar el nivell, el que haurem de fer és aconseguir passar un determinat temps a un nivell per a completar-ho. És evident que no fent res podríem doncs, passar de nivell, però indubtablement el joc no duraria gaire.

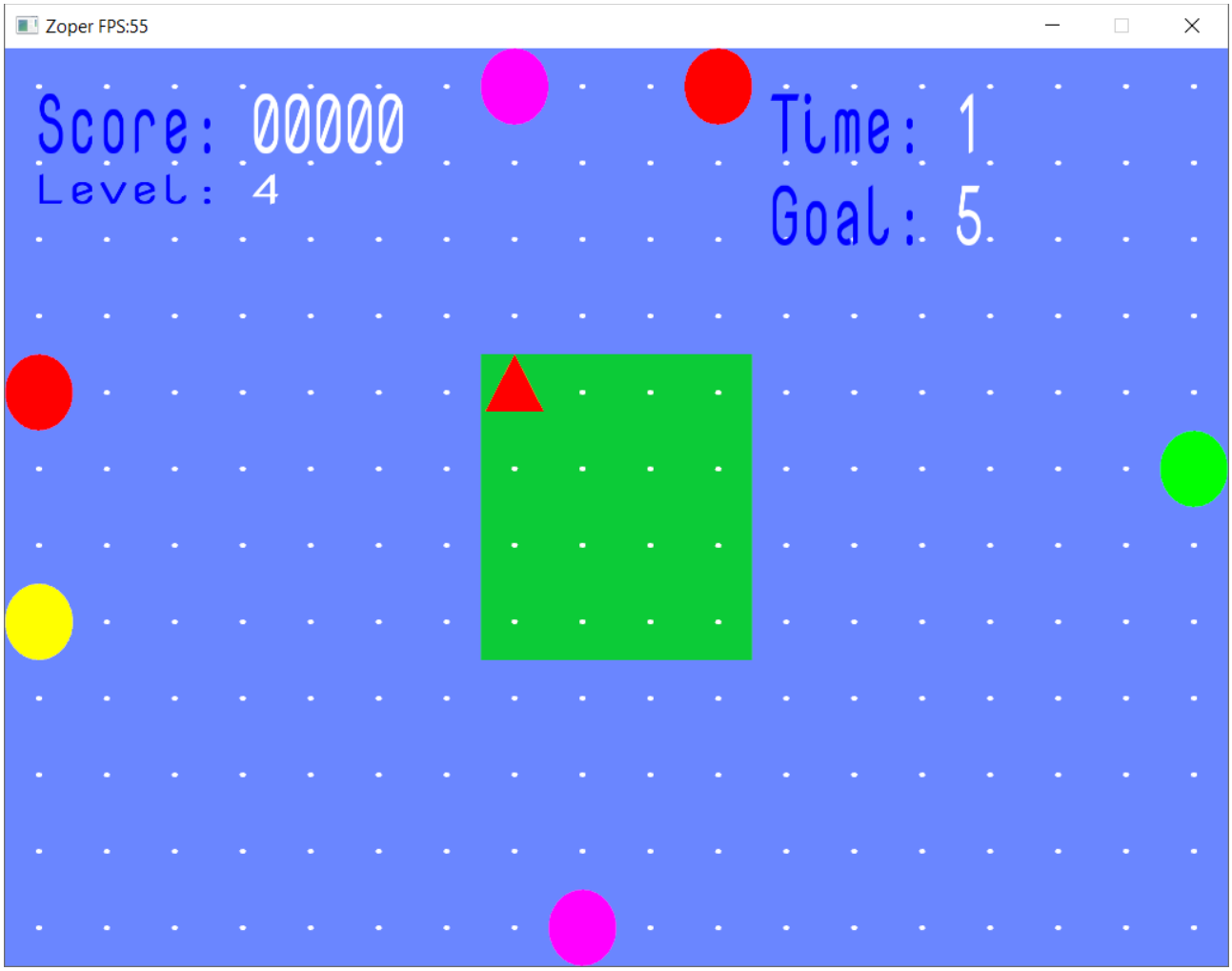


Figura 7: Jugant al mode temps

5.7 Pausa

Amb la tecla seleccionada per fer pausa, podem fer una pausa en qualsevol moment:

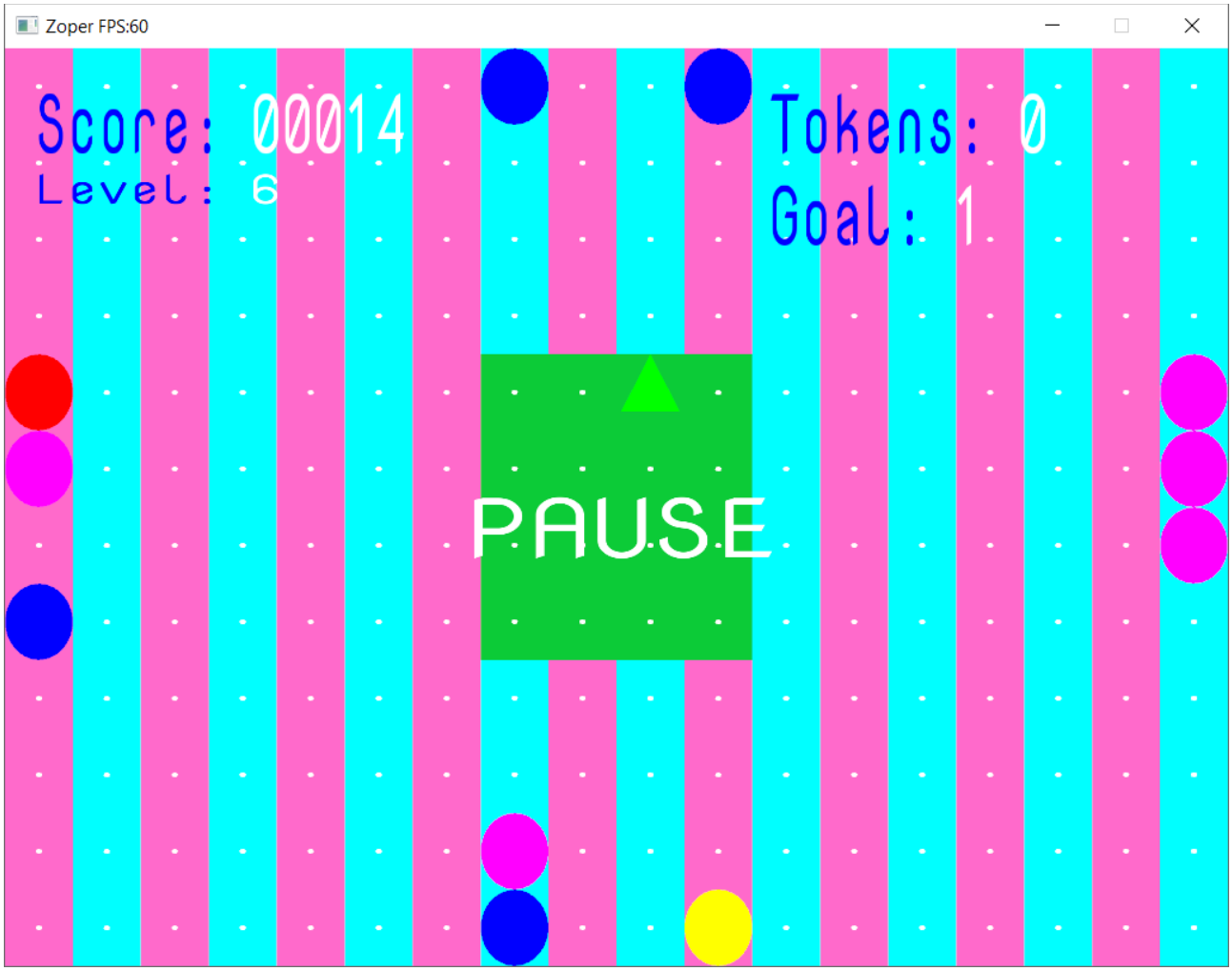


Figura 8: Pausa del joc

5.8 Final de la partida

Si alguna de les fitxes arriba al centre del taulell en qualsevol dels modes de joc, el joc s'acaba:

Figura 9: S'acabat el joc

6. Materials i eines

Per a desenvolupar aquest projecte, utilitzarem, en principi les següents eines, que poden canviar o incrementar-se durant l'execució del projecte:

- Visual Studio 2013: El projecte de codi principal serà desenvolupat utilitzant aquesta eina de Microsoft. Com és normal, l'executable generat funcionarà únicament sobre Windows. A pesar d'això, s'intentarà fer el codi el més multiplataforma possible per facilitar les conversions entre diferents sistemes.

- Biblioteca de programació gràfica SFML (Simple and Fast Multimedia Library): És una biblioteca de classes en C++ per a programar videojocs. Incorpora gràfics (shapes, sprites,…), control de finestres, àudio,… Podeu obtenir més informació de la seva pàgina web:

http://www.sfml-dev.org/

Utilitzarem la versió actual que és la 2.3.2 però podria ser actualitzada durant el desenvolupament del projecte.

- Git i github: S'utilitzarà git com a eina de control de versions i de codi. Bàsicament per la meva familiaritat amb ella i el seu potencial. El codi serà pujat i gestionat a un repositori públic a github, amb la URL:

https://github.com/LeDYoM/Zoper

en aquest repositori s'allotjà tot el codi font del programa. Encara està per decidir si també s'allotjaran recursos i gràfics.

- Webs de gràfics, fonts i músiques lliures: Depenent de les necessitats multimèdia que trobi durant el desenvolupament del projecte diferents web de recursos gratuïts per a jocs seran utilitzades. Quines seran exactament s'explicitarà més endavant.

7. Glossari

API: Application Program Interface: Interfície que proveeix una llibreria per a fer ús de si mateixa. És a dir, les funcions que exporta i que el programa client podrà utilitzar.

Classe: Element bàsic de la programació orientada a objectes. La seva representació a memòria són objectes o instàncies.

Clock: Classe sf::Clock que utilitza SFML per implementar Timers.

Git: Sistema de control de codi molt utilitzat avui en dia.

Llibreria: En el àmbit d'aquest projecte, llibreria es un fitxer o conjunt de fitxers (.dll o .so depenent del sistema operatiu) que prové d'una API o de objectes de forma externa per a la seva utilització per part d'un programa.

Memory Leak: Memòria que ha estat reservada per un programa a un punt i no alliberada. Si el programa ha de mantenir-se en execució molt temps i va requerint més memòria, al final s'esgotarà per que no es pot reservar més memòria.

Scope: Àmbit d'una variable en un programa. Les variables que no son punters estan definides normalment entre el començament i el final d'un bloc de codi (un { i un })

Render: Realitzar el dibuix d'un objecte sobre un RenderTarget. Per a fer això es calculen totes les dades necessàries, com matrius, vèrtex i colors i es fa el render.

Renderització: Acte de realitzar un render sobre alguna zona de memòria, un objecte renderitzable o la pantalla.

Resource: Pot ser un fitxer a disc amb una font o textura, però també podem parlar d'un o més bytes contenint una variable o propietat a memòria.

Timer: Classe que ajuda a controlar el temps a l'aplicació. Normalment incorpora un sistema per començar i parar. De vegades, també de pausa.

8. Bibliografia

1: http://www.stroustrup.com/C++11FAQ.html

2: https://en.wikipedia.org/wiki/Resource_Acquisition_Is_Initialization

3: http://www.sfml-dev.org/

4: http://www.bogotobogo.com/DesignPatterns/introduction.php

5: https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/Design_Patterns

6: http://en.sfml-dev.org/forums/index.php?topic=15416.0

7: https://github.com/SFML/SFML-Game-Development-Book

8: http://blog.smartbear.com/c-plus-plus/the-biggest-changes-in-c11-and-why-you-should-care/

9: https://solarianprogrammer.com/2011/12/16/cpp-11-thread-tutorial/