



Sergio Martínez Román
David Moya Gonzalez

INDEX

| | | |
|-------|--|----|
| 1. | Presentació del projecte | 4 |
| 1.1 | Descripció de l'activitat | 4 |
| 2. | Fase d'anàlisi | 5 |
| 2.1 | Anàlisi de requeriments | 5 |
| 2.2 | Estudi dels elements Hw i Sw necessaris, adaptacions requerides, Activitats del camí crític, Cost, Fases d'entrega | 5 |
| 2.2.1 | Estudi dels elements Hardware i Software necessaris | 5 |
| 2.2.2 | Planificació i gestió del temps | 6 |
| | Metodologia Agile | 6 |
| 2.2.3 | Metodologia GitFlow | 10 |
| 2.2.4 | Cost | 12 |
| 2.3 | Anàlisis i estudi del target del projecte | 12 |
| 3. | Fase de disseny | 13 |
| 3.1 | Disseny de la solució proporcionada | 13 |
| | Mockups | 13 |
| | Diagrama UML | 16 |
| 4. | Fase d'implementació | 17 |
| 4.1 | Implementació en real del projecte proposat | 17 |
| 4.1.1 | Arquitectura base implementada | 17 |
| 4.1.2 | Estats de joc | 20 |
| | Login State | 20 |
| | Register State | 20 |
| | Menú inicial State | 21 |
| | Menú opcions inicial State | 21 |
| | Menú d'opcions del joc State | 22 |
| | Menú de configuració dintre del joc | 22 |
| | Menú inventari | 23 |
| | Game State | 23 |
| | Text State | 24 |
| | Win State i Lost State | 26 |

| | | |
|-------|--|----|
| 4.1.3 | Diferents col·lisions implementades | 27 |
| 1. | Col·lisions gestionades per píxels | 27 |
| 2. | Col·lisions “efectives” i “no efectives” | 28 |
| 3. | Col·lisions rodones i quadrats | 29 |
| 4.1.4 | Musica del joc | 29 |
| 4.1.5 | Teclat implementat | 30 |
| 4.1.6 | Timer per gestió del temps de pulsació d’una tecla | 32 |
| 5. | Fase d’exploració i proves | 35 |
| 5.1 | Proves funcionals | 35 |
| | Excel de proves de col·lisions | 35 |
| 6. | Fase de manteniment i documentació del projecte | 35 |
| 1. | Manual d’usuari de l’aplicació | 35 |
| | Controls | 35 |
| | moviment | 35 |
| | Atac | 35 |
| | Tecles especials | 35 |
| | Manual d’usuari | 36 |
| | Login | 36 |
| | Iniciar Partida | 36 |
| | Moviment del personatge | 37 |
| | Atac a enemics | 37 |
| | Desactivar la música | 37 |
| | Guanyar partida | 37 |
| 7. | Valoració global del projecte | 38 |

1. Presentació del projecte

1.1 Descripció de l'activitat

Hem fet un joc 2d amb perspectiva “top-view” (vista desde adalt), sense cap mena de motor de joc, ya que el nostre objectiu era com diu el propi nom en anglés (FGD, First Game Develop) es el primer joc desenvolupat, y ho voliem fer desde l'inici tocant tots els temes que requereix un joc, y amb un llenguatge amb molt potencial de baix nivell com és el C++, només hem utilitzat una llibreria gràfica anomenada Allegro 4, que ens a aportat el poder generar una finestra per poder printar el contingut sobre ella.

En base a l'anterior mencionat, el joc amb poc temps de treball y sense experiència en el sector de desenvolupament de jocs y tampoc en totes les tecnologies utilitzades, no te cap mena d'història pero te el seu objectiu de derrotar el “boss final”, esta desenvolupat de forma que es puguin fer moltes mes funcionalitats en un futur, apart de tot això, les seves funcionalitats estan limitades com a un joc antic, esta recreat com l'ambient d'això mateix, un joc antic.

2. Fase d'anàlisi

2.1 Anàlisi de requeriments

Els requisits per al nostre joc no son molts ya que es un joc 2d que es d'un sol jugador, te les funcionalitats basiques implementades que podria tenir un joc antic.

2.2 Estudi dels elements Hw i Sw necessaris, adaptacions requerides, Activitats del camí crític, Cost, Fases d'entrega

2.2.1 Estudi dels elements Hardware i Software necessaris

Software

Per al correcte desenvolupament d'aquest joc hem tingut que instal·lar unes eines indispensables com son:

- Llibreria Allegro 4.2.2
- Compilador C++

Després com a gust nostre que també son necessaris però pots agafar altres opcions hem escollit:

- Code Blocks IDE 17.12
- MinGW (Compilador)
- Debugger GDB (Ve incluit amb el MinGW però tens que escollir quin vols)
- Git
- L'eina Kdiff3 com a solucionador de conflictes de Git
- Git Extensions
- Git Kraken

Hardware

Per al desenvolupament del nostre projecte no necessitàvem un hardware potent ya que es un joc al estil antic.

El software en aquest cas només hem tingut que tindre el Windows perque cada SO es requeria una llibreria distinta de Allegro.

2.2.2 Planificació i gestió del temps

Metodologia Agile

Nosaltres hem optat per seguir aquesta metodologia no totalment “pura” com es pot dir ya que només hem seguit part d’aquesta metodologia, com podria ser els valors que te com per exemple hem tingut com **objectiu** el software desenvolupant funcionant sobre la documentació excessiva a més de la metodologia SCRUM, Sprint, Sprint Backlog, Product Backlog.

Com explicarem després es veurà una breu explicació de com hem implementat la metodologia SCRUM completament amb exemples i tot del nostre treball, com es pot veure no hem fet tot el que la metodologia diu però ens hem guiat per aquesta, te una flexibilitat que altres no tenen.

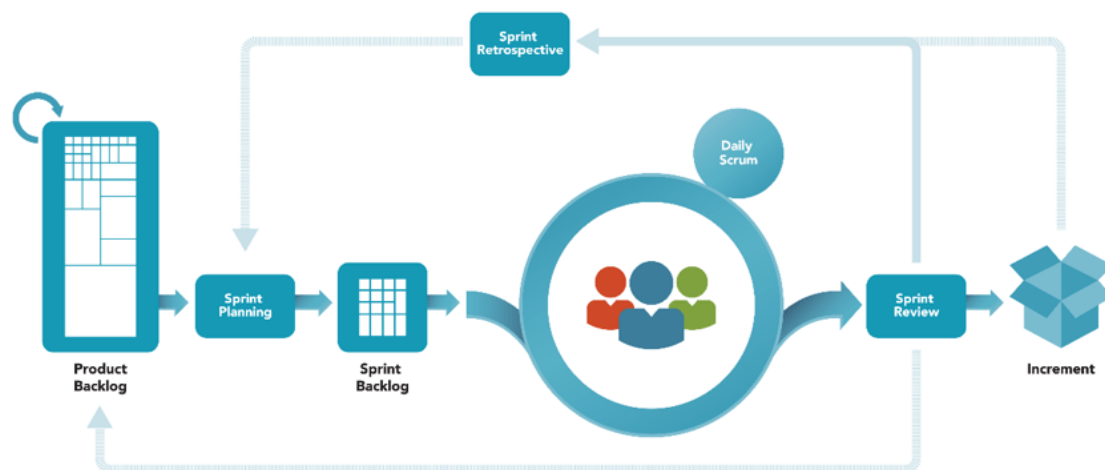
Per ordre que hem dit abans donaré una breu explicació de la implementació que hem fet sobre cada una de les metodologies seguides:

- **SCRUM:** En aquesta metodologia hem utilitzat el Trello com eina de treball per organització i planificació de temps i treball.
- **Sprint:** A medida que passaven les setmanes a cada setmana definirem un objectiu al llarg d’ella per tenir clares les prioritats.
- **Sprint Backlog:** Serien les tasques plantejades al llarg del Sprint definit en aquesta setmana
- **Product Backlog:** Nosaltres aqui hem definit les histories, o sigui, les tasques a gran escala que hem de fer durant el transcurs del desenvolupament del programari.

A més a més en la metodologia **SCRUM** també hem utilitzat el seu sistema de reunions que es molt efectiva per tenir una correcta comunicació de treball en equip i aclarir objectius que ve a ser:

- **Reunió de planificació del Sprint (Sprint Planning):** On dèiem els objectius del Sprint
- **Reunió diària (Daily Scrum):** Reunió de poc temps per dir el següent:
 - Que vam fer **ahir**.
 - Que anem a fer **avui**.
 - **Impediments** que hem tingut
- **Reunió de revisió del Sprint:** Seria bàsicament la revisió de les tasques que hem deixat en To Validate
- **Retrospectiva del Sprint (Sprint Retrospective):** Dir en que podríem millorar com equip.⁴

Aquí deixo una imatge mes aclaridora de la metodologia seguida:



En aquesta imatge es pot veure el transcurs del que seria la metodologia SCRUM que hem utilitzat, i ara en breu explicaré l'eina que hem utilitzat per la correcta funcionalitat d'aquest procediment.

Per determinar les **fases d'entrega** i el correcte funcionament de la metodologia SCRUM en el desenvolupament hem utilitzat el programari Trello.

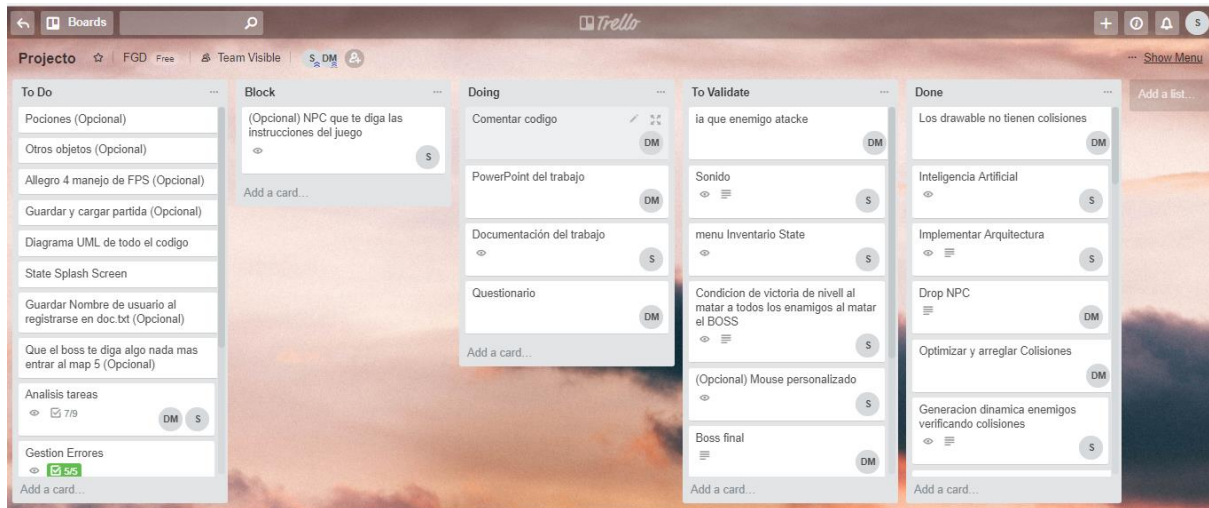


Hem definit unes columnes on van les tasques de tot el joc per fer que serien:

- To Do: Aquesta columna seria per posar les tasques que hem analitzat prèviament que estan per fer ordenades per prioritat.
- Block: Tasques que han sigut començades o no es poden fer perquè hi ha o ha sortit una tasca que tens que ferla per posteriorment terminar aquesta tasca de la columna Block.
- Doing: Tasques que estem desenvolupant actualment.
- To Validate: Tasques que n'hem traslladem de Doing a aquesta al finalitzar-la baix el nostre propi criteri (o sigui qui la desenvolupat).
- Done: Les tasques que hem validat entre els dos que es trobaven en la columna To Validate.

Aquest programari o altres similars a aquests son ideals no només per al treball en equip, inclòs per gestionar el teu propi treball, a més que aquest en específic Trello es gratis i compte amb totes les eines necessàries per gestionar un projecte.

Exemple de la nostra gestió en el Trello:



Com es pot veure a la imatge s'aprecia les tasques més importants ja estan en Done ja que li hem donat la prioritat adequada segons el nostre criteri.

Les nostres **prioritats** davant d'aquest projecte a sigut:

- **L'Arquitectura** del joc que es la base de tot.
- **La funcionalitat** del joc de més bàsiques a més avançades.

En el nostre cas ens ha faltat una altra taula per al nostre gust que es digués Backlog per definir tasques que ens anés sortint o pensant però que no definiríem en el nostre Sprint.

Però ja que era un projecte petit de només 1 mes hem decidit prescindir d'aquesta opció.

2.2.3 Metodologia GitFlow

Aquesta metodologia que hem seguit es per implementar un flux de treball entre els dos en aquest cas que serveix per facilitar la gestió de treball d'una forma organitzada.

Consisteix en crear una “branca” teva de codi per fer el:

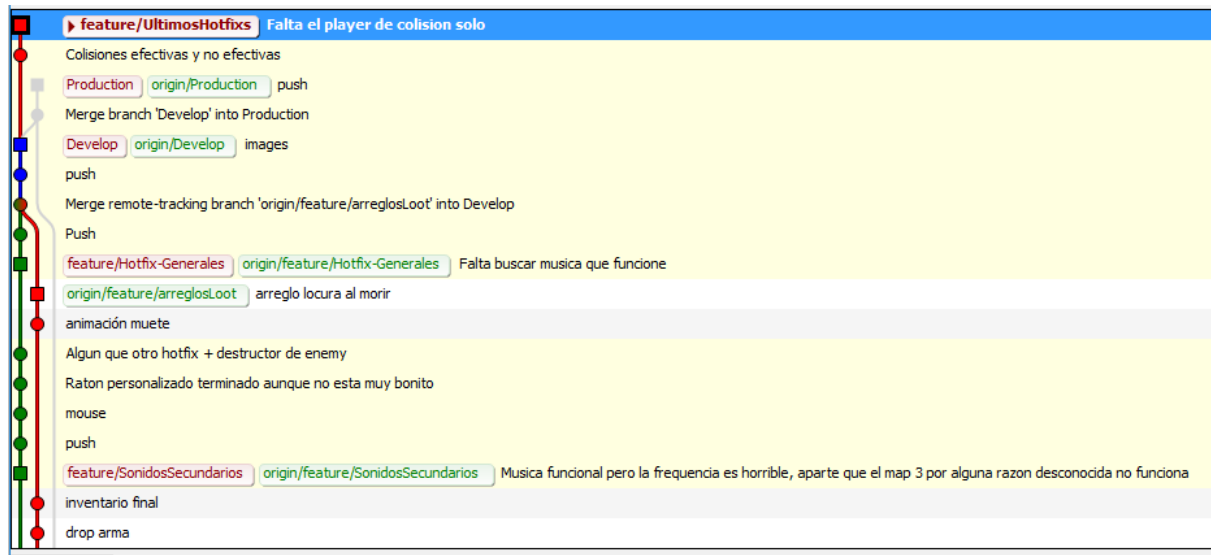
- **Feature:** Funcionalitat que vas a implementar.
- **Hotfix:** Arreglar una funcionalitat del programari.

Encara que també hi ha la rama **Release** nosaltres només hem implementat aquestes ya que no requeríem de mes necessitats.

Hem subdividit el nostre treball en dos rames principals que ve eren “**Develop**” i “**Production**”.

- **Develop** aquí desenvolupem el programari i a medida que anem sortint d'aquesta branca per fer funcionalitats o hotfixs com abants hem explicat juntem el codi en Develop que estaria per Validar, tal i com hem senyalat en el Trello.
- **Production** aquesta seria una branca en la qual es pujaria el codi que si o si ya esta validat i comprovat que funciona correctament i totalment terminat.

Aquí deixo una imatge d'exemple de un petit flux nostre de treball que hem fet en l'eina **GitExtensions** o **GitKraken** que es el software que ens a facilitat el treballar aquesta metodologia:



Com podeu apreciar es veu la branca d'on surt on torna a entrar i el missatge al guardar la branca en local o origin.



Com a repositori per guardar el nostre codi de git hem utilitzat el **GitHub**, es un repositori públic que es podrà visualitzar tot el nostre treball, inclòs hem posat un petit manual i aquesta documentació perquè si alguna persona vol aprendre a desenvolupar un joc pot començar per aquí i donar-se una idea de com fer-ho.

Aquí la imatge del nostre repositori amb tot el codi pujat.

Projecto final curso

allegro4 2dgame cpp codeblocks-ide

239 commits 53 branches 0 releases 2 contributors

Branch: Develop New pull request Create new file Upload files Find file Clone or download

| | |
|----------------------|--|
| Sergio225 images | Latest commit a4e3c3a 10 hours ago |
| Api | New Changes 2 months ago |
| Diagrama UML | Diagrama UML generado a code a month ago |
| Documentació | Images 10 hours ago |
| FGD | Images 10 hours ago |
| Otros proyectos test | 03/05 29 days ago |
| Test | Funcional / Solo estan las declaraciones de las matrices falta la imp... 24 days ago |
| resourcesFGD | BMP5 BOSS 3 days ago |
| .gitignore | El gitignore liante 16 days ago |
| Aquest | Donar en codi 24 days ago |

2.2.4 Cost

Es un joc gratis ya que el nostre propòsit era desenvolupar el primer joc que hem fet, com a molt et podria costar la llicència de Windows.

2.3 Anàlisis i estudi del target del projecte

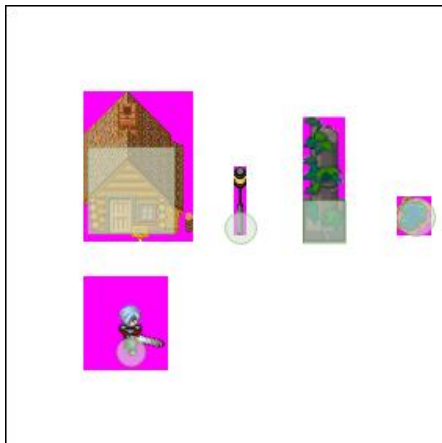
Aquest joc esta destinat a persones de pc, es pot llançar el programa en la plataforma Steam que una quantitat alta d'usuaris que els hi podria interessar el nostre joc.

3. Fase de disseny

3.1 Disseny de la solució proporcionada

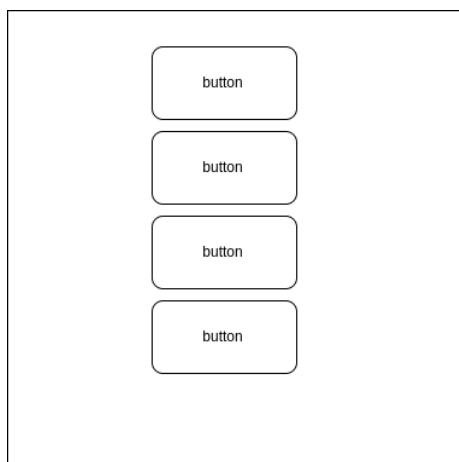
Mockups

Hem realitzat una sèrie de mockups per el anàlisis abans de implementar-lo en el nostre joc, molts ho hem fet a mà ya que es el mes ràpid i mes eficient però altres ho hem fet amb programari.

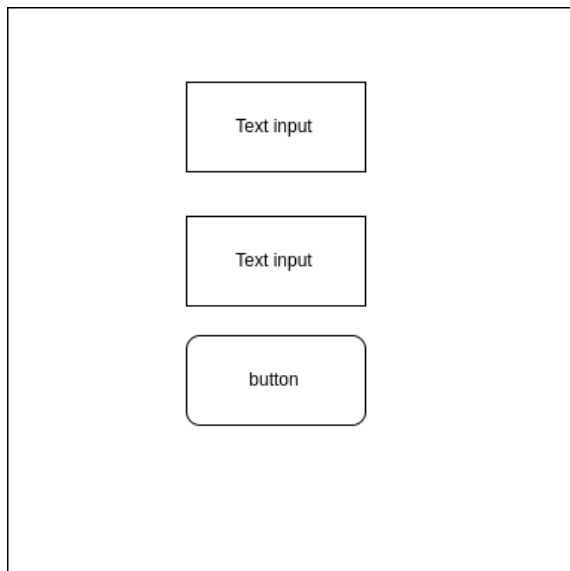


El disseny de les col·lisions que seguiran els nostres objectes.

Aquí presentem el mockup del State de la majoria dels menús principal del joc.

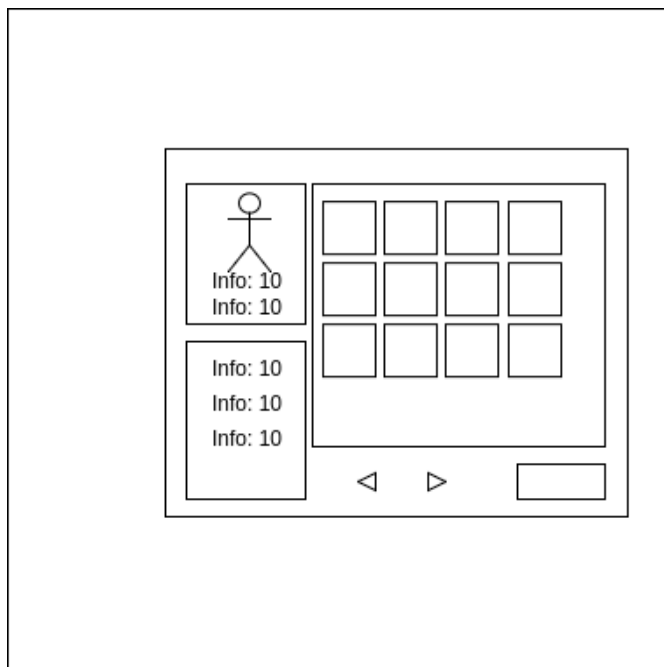


Mockup del Login



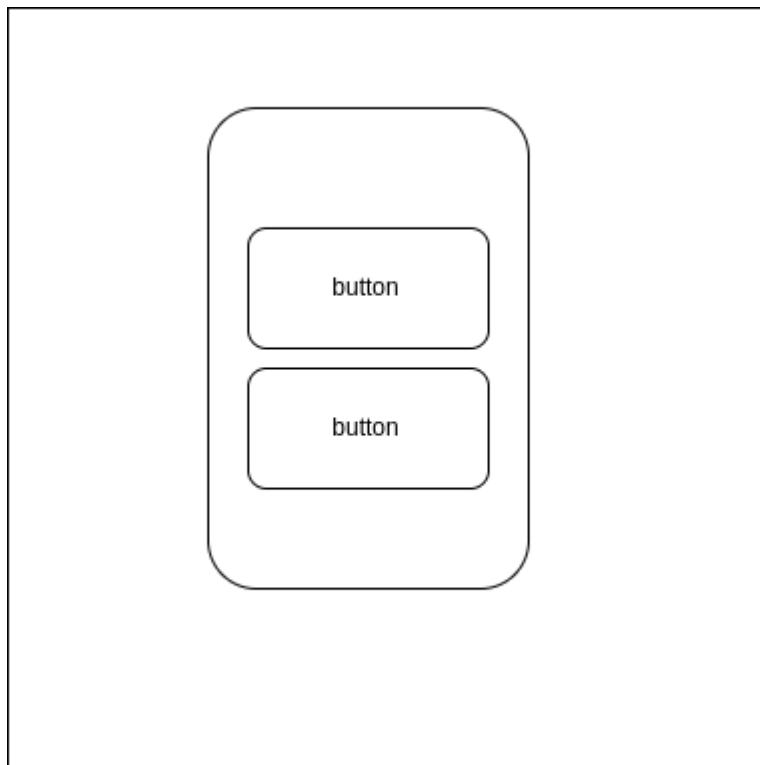
A mockup of a login screen. It consists of a large rectangular container. Inside, there are three elements stacked vertically: a rectangular text input field with the text "Text input", another rectangular text input field with the text "Text input", and a rounded rectangular button with the text "button".

Mockup del inventario del player dins del joc



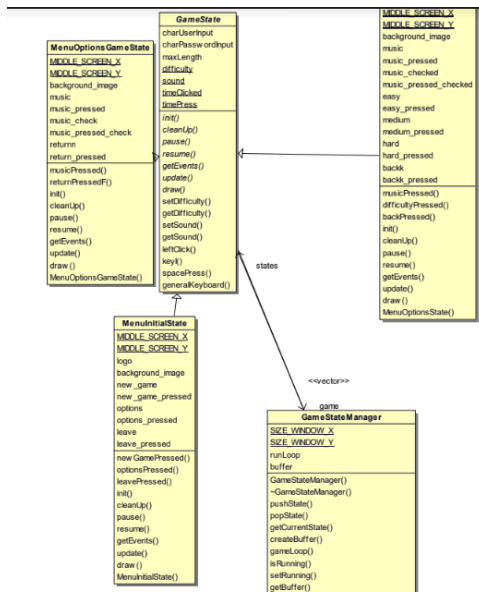
A mockup of a player's inventory screen. It features a large rectangular container. Inside, there is a smaller rectangular frame. On the left side of this frame, there is a vertical stack of three boxes. The top box contains a stick figure icon and the text "Info: 10" twice. The bottom two boxes each contain the text "Info: 10" three times. To the right of these boxes is a 3x4 grid of 12 empty square slots. At the bottom of the frame, there are three elements: a left-pointing triangle, a right-pointing triangle, and a small rectangular input field.

Mockup del menú d'opcions del joc al donar-li a "Esc"



Altres mockups que no mostrem aquí estan realitzats a mà per nosaltres mateixos per seguir una planificació abans de la implementació.

Diagrama UML



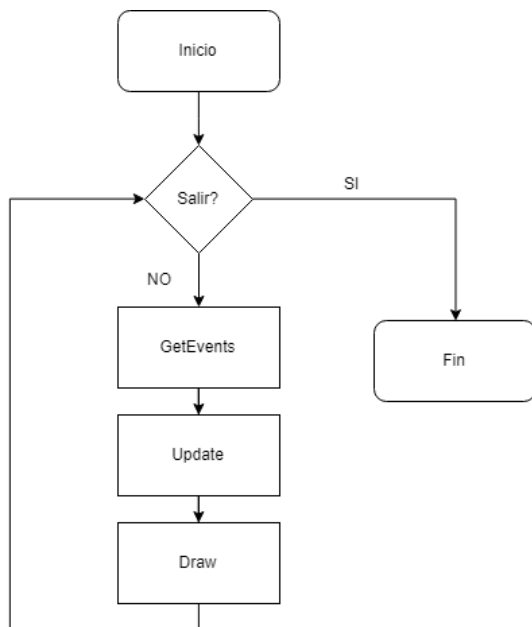
Aquest es un petit diagrama UML de part del nostre codi implementat per l'arquitectura desenvolupada que després explicarem en que consisteix.

4. Fase d'implementació

4.1 Implementació en real del projecte proposat

4.1.1 Arquitectura base implementada

El nostre joc consta d'una arquitectura prèviament analitzada i ben organitzada la qual es la base del joc, es qui gestiona el bucle principal que te un joc que bàsicament el bucle seria així:



Com es pot veure un joc es basa sempre en el mateix bucle principal **abans d'això** primer el joc s'inicialitza e instal·les o prepares els requeriments que necessiti el teu joc abans d'entrar al bucle, una vegada iniciat el bucle seria exactament el que mostrem a continuació per ordre del bucle:

1. **GetEvents:** reconeixes de l'usuari events que ell faci amb la interacció amb el joc per exemple el donar-li a la tecla "Esc".
2. **Update:** en aquesta iteració actualitzaríem les dades del joc, ya que un joc no para de passar coses tal y com un personatge estigui caminant, això seria traduït en aquesta iteració com que aquest personatge estigues augmentant la seva posició X o Y.
3. **Draw:** aquí bàsicament seria després de recollir els events i les actualitzacions de les dades el printar totes aquestes dades a sobre la nostra finestra.

Una vegada explicat s'entén resumint que un joc es una repetició continua de actualitzacions i printar per pantalla.

La nostra arquitectura principal bàsicament es basa en la mencionada anteriorment com qualsevol joc com hem dit, llavors aquesta seria la primera part de l'arquitectura que hem implementat.

La segona part d'aquesta **arquitectura** que hem implementat la hem triat segons els nostres requeriments i necessitats com a joc 2d i llenguatge de programació orientada a objectes C++, nosaltres com mostrarem a la següent imatge tenim una interfaç per a tots els nostres estats de joc que explicarem després quins tenim i perque.

```
class GameState {  
  
public:  
  
    GameStateManager *game;  
    Music managerMusic = Music();  
    Player player;  
  
    /**  
    Initial methods  
    */  
    virtual void init()=0;  
    virtual void cleanUp()=0;  
  
    virtual void pause()=0;  
    virtual void resume()=0;  
  
    /**  
    Methods Loop  
    */  
    virtual void getEvents()=0;  
    virtual void update()=0;  
    virtual void draw()=0;
```

Com es pot veure tenim unes funcions que utilitzarà tots i cada un dels nostres estats obligatòriament, aquest punt d'obligatori el definim nosaltres per evitar errors, només hem tingut que fer una interfaç pura (per definir una interfaç pura es simplement una sintaxis que necessita C++ sobre la funció que vols fer pura, que farà que les classes derivades tinguin que implementar la si o si).

Ara procedirem a explicar cadascuna d'aquestes funcions que hem definit segons el nostre criteri.

- **Init:** Serà la inicialització de dades del stat que es vagi a posar en curs.
- **CleanUp:** S'utilitza per netejar dades inservibles o lliurar memòria dels punters per gestionar el rendiment del programari.
- **Pause:** El codi que s'executarà quan el state actual quedi fora de la finestra mostrada.
- **Resume:** El mateix que el pause però es quan torna a mostrar-se per pantalla.
- **GetEvents:** Les actualitzacions d'entrada que rebeix de l'usuari i com interactuarà el programari amb ell.
- **Update:** Actualitzacions del joc que estarà en transcurs independentment del stat en el que s'estigui.
- **Draw:** Printa tot el contingut del stat actual per pantalla.

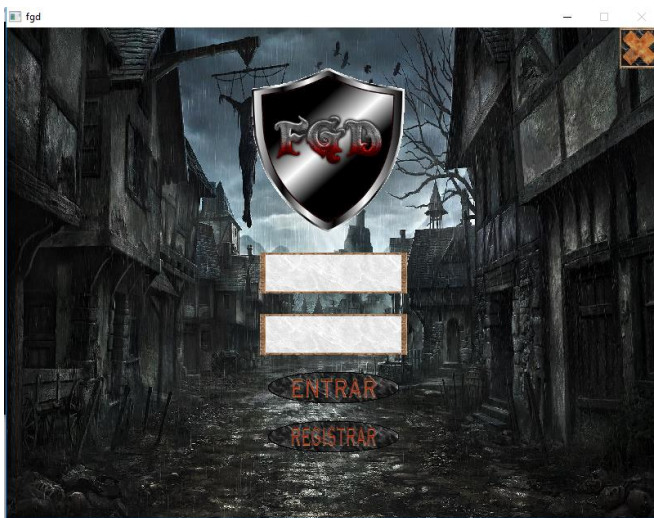
Amb tota l'**arquitectura** preparada tal i com hem explicat que hem fet, ara existirà una navegació entre estats de joc perfectament controlada i organitzada, es un punt molt important que si no implementes correctament mes endavant si es un projecte gran o petit inclòs et veuràs en problemes de no haver-ho fet, ya que tot es farà mes complexe i no sabràs on trobar el teu propi codi.

4.1.2 Estats de joc

Els estats de joc seria on et trobes en aquest moment en el joc (per exemple el menú, opcions del joc, el joc en sí mateix, el inventari inclòs), tot el que es mostra per pantalla es un estat distint.

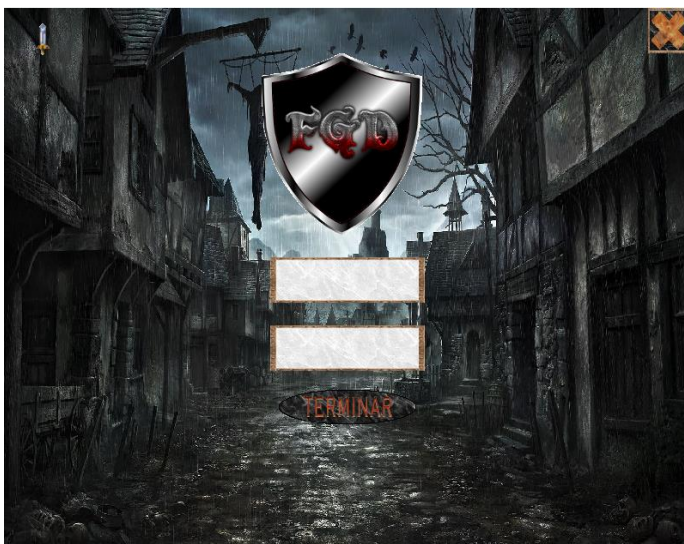
Llavors en base als mockups realitzats i altres proves que no estan documentades que hem fet o pensat, ha sortit estats del joc com son el que es visualitzarà a continuació:

Login State



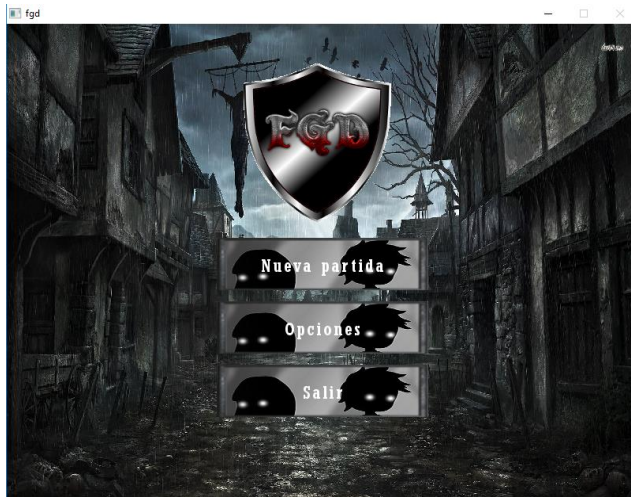
En aquest estat es pot apreciar com pots Entrar amb el teu Usuari en el seu defecte et pots Registrar, tens la creu per poder sortir ràpidament de l'aplicació, i en aquest cas al no utilitzar un motor de joc hem tingut que implementar un teclat personalitzat amb cada tecla numerada y cada una guarda el valor corresponent.

Register State



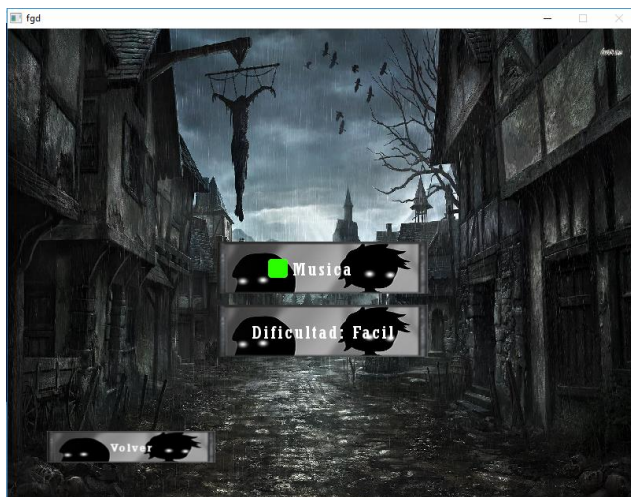
Bàsicament seria el mateix que el Login sols que tens el botó per terminar el registre o mateix es per tornar enrere i no registrar-te.

Menú inicial State



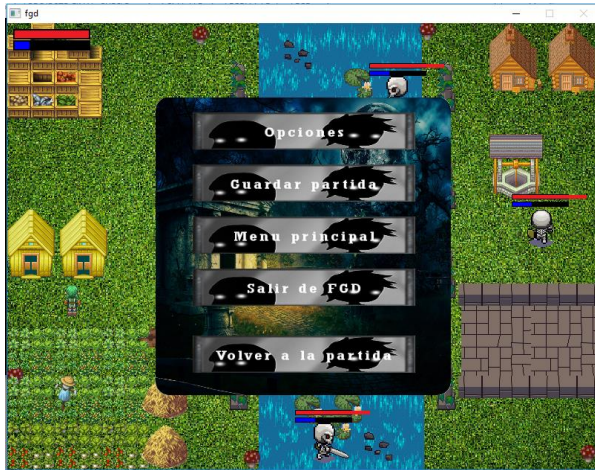
El següent estat es el menú inicial del joc on pots Iniciar una nova partida, anar a les opcions del joc on en un futur implementarem més funcionalitats a aquest, y un botó per sortir de l'aplicació (no hem utilitzat la creu perquè hem vist oportú utilitzar-la només en el Login).

Menú opcions inicial State



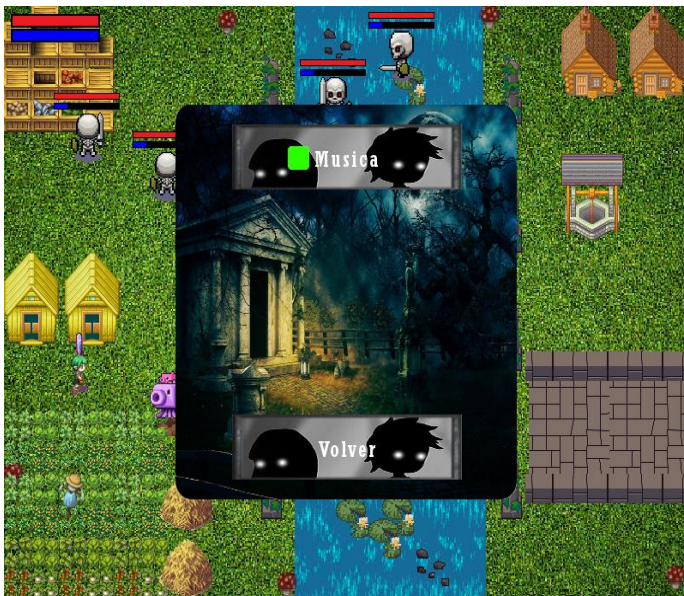
Aquí tenim el menú d'opcions principal on només hem implementat 2 opcions per falta de temps i hem preferit donar prioritat a altres funcionalitats del joc que hem vist més importants.

Menú d'opcions del joc State



També el joc te el seu menú d'opcions on pots entrar a la configuració com al menú inicial d'opcions (l'anterior captura) on trobaràs només el poder desactivar o activar la musica, també tens accés a guardar la partida, i 3 accessos ràpids o al menú principal, o a sortir del joc de cop o tornar a la partida.

Menú de configuració dintre del joc



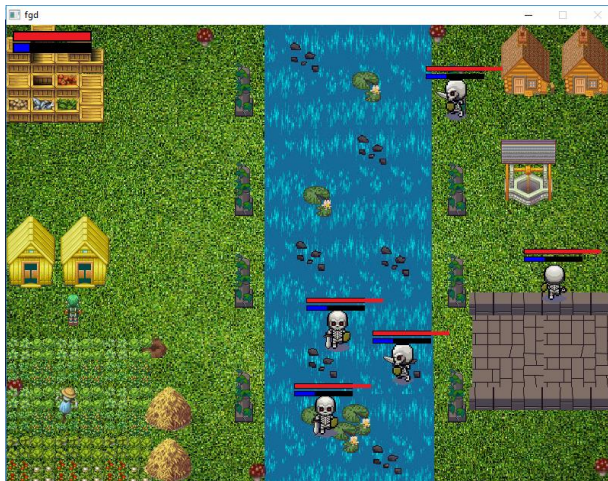
Aquí seria les opcions dintre del joc quan anteriorment li dones al botó d'opcions, encara que pots implementar altres funcionalitats, ara mateix només n'hi ha la de Musica per desactivar o activar la de tot el joc.

Menú inventari



El menú inventari permet mostrar la informació del personatge a més de la quantitat d'objectes en el inventari. En pressionar sobre un objecte de l'inventari mostra la seva informació i en cas de ser un arma el selecciona com arma principal.

Game State



Aquí tenim el game state que ell mateix gestionaria ell aniria canviant entre els diferents mapes que hem dissenyat, i aquí mostrem el primer disseny dels mapes que tenim podem observar que hi poden haver varis tipus de col·lisions com son rodones contra quadrats, rodones amb rodones, quadrats contra quadrats, totes estan gestionades per nosaltres mateixos ya que la

llibreria que hem utilitzat no ens proporcionava cap mena d'ajuda respecte això.

Text State

Aquest State es una classe definida que segons el requeriment li passarem un paràmetre perquè es mostri un text a una zona del mapa o un altre, seria com quan comencem el joc ens parla un NPC que ens diu les instruccions del joc seria així:



Aquest es un exemple de com es mostraria, nosaltres li diem que estem al primer mapa llavors ell executa l'ordre determinada a aquest paràmetre.

```
TextState::TextState(GameStateManager *game, int callState)
{
    this->game = game;
    //El estado que tendra esta clase al llamarse segun que integer
    this->callState = callState;

    this->init();
}

void TextState::init()
{
    if(this->callState == 0){
        this->managerMusic.soundMap1();
    }

    this->checkFirstText = false;
    this->nextText = 1;
    this->secondText = 0;
    this->checkSecondText = false;
    this->checkNextText = false;
    this->vectorBocadillo1Mapa.push_back("Pressiona el espai");
    this->vectorBocadillo1Mapa.push_back("Primer texto");
    this->vectorBocadillo1Mapa.push_back("Segundo texto");
    this->vectorBocadillo1Mapa.push_back("Tercer texto");
    this->vectorBocadillo1Mapa.push_back("Cuarto texto");
}
```

Aquí veiem que rebeix el paràmetre (callState), i col·loquem el text a mà en un vector de punters “char”, que gestionem de la segona manera:


```

if(this->checkNextText){
    textout_ex(this->game->getBuffer(), font, this->vectorBocadillo1Mapa.at(nextText),
        90, 280, makecol(0, 0, 0), -1);
}
if(this->checkSecondText){
    textout_ex(this->game->getBuffer(), font, this->vectorBocadillo1Mapa.at(secondText),
        90, 290, makecol(0, 0, 0), -1);
}

if(GameState::spacePress()){
    if(this->nextText >= this->vectorBocadillo1Mapa.size()){
        this->game->popState();
    }
    if(!this->checkNextText){
        this->secondText+=2;
        this->checkNextText = true;
        this->checkSecondText = false;

        //El primer espacio ya borrara el primer text
        this->checkFirstText = true;
    }else if (this->checkNextText && !this->checkSecondText) {
        if(this->secondText >= this->vectorBocadillo1Mapa.size()){
            this->managerMusic.stopSoundMap1();
            this->game->popState();
        }
        this->checkSecondText = true;
    }else{
        this->nextText+=2;
        this->checkNextText = false;
        this->checkSecondText = false;
    }
}
}

```

Tornem a veure que tota la gestió es a mà, en aquest àmbit del programari esta gestionada per píxels on te que printar i el que.

Es un algoritme que el que fa es:

1. Pressiones l'espai per passar de text
2. Això fa que el primer text que esta apartat s'esborri i comenci el "bucle" entre el vector segons les posicions que hem assignat a "nextText" i "secondText"
3. Primer es mostrarà el nextText que estarà posicionat després del firstText
4. Si tornes a donar-li a l'espai mostrarà el secondText
5. Tot això es basa en anar sumant de 2 en 2 a cada un dels índex per mostrar correctament i a la vegada dintre del cercle mostrat per pantalla

Win State i Lost State



Aquest es el game over State que s'activarà òbviament quan perdis, et donarà les opcions de tornar al menú, carregar la partida per tornar a un altre punt anterior e intentar-ho de nou, i l'opció de sortir del joc sencer.



Aquest el Win State que s'activarà quan derrotris al enemic final i traspasar la porta de l'últim mapa.

Aquí només et donarà l'opció de sortir al menú o sortir del joc, ya que es suposa que al guanyar no vols tornar enrere.

4.1.3 Diferents col·lisions implementades

Actualment disposem de 5 mapes (dic actualment per si en un futur actualitzem el joc a altre versió), en la implementació de les col·lisions en tots els mapes estan gestionades de distintes formes que en el nostre projecte hem fet la **gestió de les col·lisions** següents:

- Col·lisions rodones amb rodones
- Col·lisions rodones amb quadrats
- Col·lisions quadrats amb quadrats
- Col·lisions per píxels
- Col·lisions efectives i no efectives

1. Col·lisions gestionades per píxels

Les col·lisions gestionades per píxels les hem tingut que implementar per el simple motiu que al fer el joc d'aquesta manera (com si fos un joc antic sense motors gràfics potents com els que hi ha ara), la imatge de fons que tindria el mapa si te un objecte amb el qual podria col·lisionar el player o enemic no el puguem utilitzar perquè ve amb la imatge inclusiva no es un objecte que hem col·locat amb la programació orientada a objectes, llavors et veus obligat a gestionar la col·lisió agafant on estant els píxels d'aquest objecte, també dir que l'utilitzem per als límits de la finestra del joc en tots els mapes.

Un exemple de com gestionem les **col·lisions gestionades per píxels** a l'últim mapa que tenim seria el següent:

```
if(this->player.getX() <= 24) this->player.setXandAX(this->player.getAX());
if(this->player.getX() >= 703) this->player.setXandAX(this->player.getAX());
if(this->player.getY() <= 49) {
    if(this->player.getX() <= 334 || this->player.getX() >= 392){
        this->player.setYandAY(this->player.getAY());
        this->player.setXandAX(this->player.getAX());
    }
}

if(this->player.getY() >= 420){
    if(this->player.getX() <= 290 || this->player.getX() >= 434){
        this->player.setYandAY(this->player.getAY());
        this->player.setXandAX(this->player.getAX());
    }
}
```

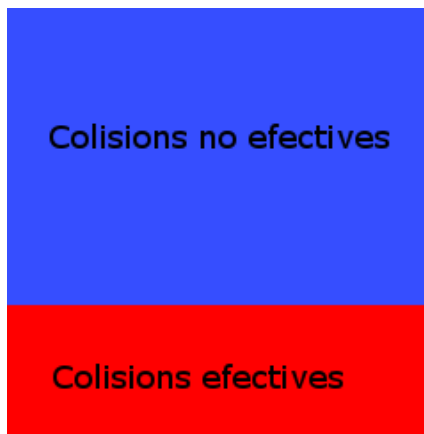
Aquí resumint el que estem fent es agafar les posicions del player X i Y, en base a les dades que ens proporciona aquests getters si determinen que estan fora del límit li torna a col·locar la posició a l'anterior abans de sortir-se.

2. Col·lisions “efectives” i “no efectives”

L'anomenem així per dir-ho d'alguna manera, l'objectiu d'aquesta gestió es per si un player o enemic passa per enrere d'una casa o el que haguem determinat es visualitzi com si la casa estigués per a sobre d'aquest, i si passa per endavant la casa quedi per enrere de la imatge d'aquest player o enemic, i la explicació per les col·lisions efectives o no efectives la explicació a aquestes seria el següent:

- Col·lisió efectiva es un radi que li col·loquem al objecte que significa es que si el player/enemic que col·lisió amb aquest objecte li col·locarà la posició anterior.
- Col·lisió no efectives es el radi restant al que queda de la imatge que no hem col·locat de la col·lisió efectiva, per que si el player o enemic passa per la zona de col·lisió no efectiva la imatge de la casa es printara sobre aquest personatge.

Imatge de l'explicació del que seria aquestes dues col·lisions:



El quadrat en sí seria la imatge que col·locaries al teu joc i aquest seria un exemple del radi que col·locaries a cada tipus de col·lisió a la teva imatge.

3. Col·lisions rodones i quadrats

Son el tipus de col·lisió amb les quals es basen totes les col·lisions que hem fet **menys** les col·lisions per píxels.

El que fem aquí es col·locar a cada objecte el tipus de col·lisió que tindrà en el seu constructor (rodona o quadrada), amb aquesta dada ya després gestionem aquesta segons els seus requeriments com podrien ser:

- Player amb l'enemic (rodona amb rodona)
- Player amb ambient (en aquest cas podria ser tots els tipus de col·lisions abans mencionades)
- Enemic amb enemic (rodona amb rodona)

4.1.4 Musica del joc

Per fer la implementació de la musica del joc hem creat una classe nova perquè segons el map que s'està mostrant actualment s'escolti una o altre musica, també cada vegada que ataca o rebeix dany el personatge s'escolta un altre efecte.

Aquí tenim la implementació de la classe Musica que hem creat:

```
#include <vector>

class Music {

public:
    Music();
    void init();

    /**
     * Sonidos para reproducir
     */
    void soundMenu();
    static bool isPlayingMenu;
    void soundAttack();
    void soundMap1();
    static bool isPlayingMap1;
    void soundMap2();
    static bool isPlayingMap2;
    void soundMap3();
    static bool isPlayingMap3;
    void soundMap4();
    static bool isPlayingMap4;
    void soundMap5();
    static bool isPlayingMap5;
    void soundWin();
    static bool isPlayingWin;
    void soundLost();
    static bool isPlayingLost;
    //void soundWounded();

    void stopSoundMenu();
    void stopSoundMap1();
```

S'aprecia que tot esta gestionat per nosaltres mateixos perquè la mateixa llibreria Allegro tampoc ens proporcionava ayuda en això, només en poder reproduir-la.

Els mètodes que teníem que implementar per cada musica o efecte era:

- **Parar** la musica o efecte
- **Reproduir** la musica o efecte
- **Booleana** per conèixer el seu estat

4.1.5 Teclat implementat

Hem tingut que implementar un teclat general per tota l'aplicació i tecla per tecla guardem en un punter tipus “char” a la vegada que col·locàvem una booleana per gestionar si hem pressionat o no una tecla, igualment millor que l'explicació deixo una imatge que acarrà tot.

```
void GameState::generalKeyboard(int userOrPass)
{
    char *characterInput = new char();
    bool checkPress = false;
    if (Timer::getTime() - 5 > this->timePress) {

        if (key[KEY_A]) {characterInput = "a"; checkPress = true;}
        if (key[KEY_B]) {characterInput = "b"; checkPress = true;}
        if (key[KEY_C]) {characterInput = "c"; checkPress = true;}
        if (key[KEY_D]) {characterInput = "d"; checkPress = true;}
        if (key[KEY_E]) {characterInput = "e"; checkPress = true;}
        if (key[KEY_F]) {characterInput = "f"; checkPress = true;}
        if (key[KEY_G]) {characterInput = "g"; checkPress = true;}
        if (key[KEY_H]) {characterInput = "h"; checkPress = true;}
        if (key[KEY_I]) {characterInput = "i"; checkPress = true;}
        if (key[KEY_J]) {characterInput = "j"; checkPress = true;}
        if (key[KEY_K]) {characterInput = "k"; checkPress = true;}
        if (key[KEY_L]) {characterInput = "l"; checkPress = true;}
        if (key[KEY_M]) {characterInput = "m"; checkPress = true;}
        if (key[KEY_N]) {characterInput = "n"; checkPress = true;}
        if (key[KEY_O]) {characterInput = "o"; checkPress = true;}
        if (key[KEY_P]) {characterInput = "p"; checkPress = true;}
        if (key[KEY_Q]) {characterInput = "q"; checkPress = true;}
        if (key[KEY_R]) {characterInput = "r"; checkPress = true;}
        if (key[KEY_S]) {characterInput = "s"; checkPress = true;}
        if (key[KEY_T]) {characterInput = "t"; checkPress = true;}
        if (key[KEY_U]) {characterInput = "u"; checkPress = true;}
        if (key[KEY_V]) {characterInput = "v"; checkPress = true;}

        if (key[KEY_8]) {characterInput = "8"; checkPress = true;}
        if (key[KEY_9]) {characterInput = "9"; checkPress = true;}

        if (key[KEY_BACKSPACE]) {
            if (userOrPass == 1) {
                if (this->charUserInput->size() > 0) {
                    this->charUserInput->pop_back();
                }
            } else if (userOrPass == 2) {
                if (this->charPasswordInput->size() > 0) {
                    this->charPasswordInput->pop_back();
                }
            }
        } else if (userOrPass == 1 && checkPress && this->charUserInput->size() < this->maxLength) {
            this->charUserInput->push_back(characterInput);
        } else if (userOrPass == 2 && checkPress && this->charPasswordInput->size() < this->maxLength) {
            this->charPasswordInput->push_back(characterInput);
        }

        this->timePress = Timer::getTime();
    }
}
```

Aquí tenim la implementació nostra del teclat, amb totes les tecles que venen a ser lletres i números, com podeu veure el que mes por confondre seria, el per que serveix el “UserOrPass”, que bàsicament seria en el State Login:



Segons el paràmetre que li passem sabrem si estem escrivint a sobre del quadre User o el quadre Pass.

Com hem vist a la imatge de la implementació del teclat guardem cada lletra a un vector, cada quadre de text té un vector que conté punters a tipus “char”, llavors en base al paràmetre proporcionat al mètode del teclat, podem determinar en quin vector tenim que esborrar o escriure la lletra pressionada.

4.1.6 Timer per gestió del temps de pulsació d'una tecla

El timer com diu el títol d'aquest sub-apartat, es una classe que hem implementat amb l'objectiu de gestionar el temps entre pulsació entre tecla i tecla o inclòs la pressió sobre un click del ratolí.

La pregunta del perquè s'hi ha de fer això es per el simple motiu de que un joc com havíem dit es un simple bucle que actualitza i printa per pantalla, llavors aquest bucle s'executa cada un determinat temps que hem definit nosaltres, que ve a ser cada 10 milisegons.

```
void GameStateManager::gameLoop()
{
    while(this->isRunning()) {
        Timer::timerTIC();

        this->getCurrentState()->getEvents();

        this->getCurrentState()->update();

        this->getCurrentState()->draw();

        rest(10);
    }
}
```

El rest es la definició del temps el qual descansa el bucle.

Una vegada sabem que s'executa cada tan poc temps, pots veure el perquè d'aquesta classe, si tu pressiones la lletra "a" en el camp de text d'usuari del Login State no et dona temps humà a soltar la tecla llavors hem implementat la classe Timer (anomenada per nosaltres), aquí teniu la definició:

```
#ifndef Timer_h
#define Timer_h

class Timer {
private:
    static int time;
public:
    static void timerTIC();
    static int getTime();
};

#endif //Timer_h
```

Bàsicament es basa en 2 mètodes i un integer.

- **Integer time** seria el temps transcorregut des de l'última pulsació
- **Mètode timerTic** es per augmentar el temps d'un en un.
- **getTime** un simple getter del time.

```
#include "Timer.h"

int Timer::time = 0;

void Timer::timerTIC() {
    time++;
}

int Timer::getTime() {
    return time;
}
```

Aquí teniu la implementació que es tal i com hem dit a l'explicació.

Ara col·locarem un exemple d'ús d'aquesta classe que seria el següent:

```
bool GameState::spacePress()  
{  
    if((key[KEY_SPACE]) && (Timer::getTime()-10 > this->timePress))  
    {  
        this->timePress = Timer::getTime();  
        return true;  
    }  
    return false;  
}
```

Aquí com podeu veure es un exemple d'ús de la classe on col·loquem el temps de pulsació per la tecla “Espai”.

El que retorna aquest mètode es una booleana dient si pots o no tornar a polsar la tecla espai, quan sigui verdader significa que la pots tornar a polsar.

5. Fase d'exploració i proves

5.1 Proves funcionals

| | Step Action | | | | Expected Result | Result | |
|----|--|---|---|----|---|--------|--|
| P1 | Colision circular con circular | 0 | 0 | P1 | Devuelve true dado que ha colisionado | OK | |
| P1 | Colision circular con circular | 0 | 0 | P1 | Devuelve false dado que no ha colisionado | OK | |
| P2 | Colision circular con cuadrada efectiva | 0 | 0 | P2 | Devuelve true dado que ha colisionado | OK | |
| P2 | Colision circular con cuadrada efectiva | 0 | 0 | P2 | Devuelve false dado que no ha colisionado | OK | |
| P3 | Colision circular con cuadrada no efectiva | 0 | 0 | P3 | Devuelve true dado que ha colisionado | OK | |
| P3 | Colision circular con cuadrada no efectiva | 0 | 0 | P3 | Devuelve false dado que no ha colisionado | OK | |

Aquí tenim un seguiment de proves a realitzar per l'usuari per provar les col·lisions.

6. Fase de manteniment i documentació del projecte

1. Manual d'usuari de l'aplicació

Controls

moviment

- Fletxa amunt: moure personatge cap amunt
- Fletxa dreta: moure personatge a la dreta
- Fletxa abaix: moure personatge cap a baix
- Fletxa esquerra: moure personatge a l'esquerra

Atac

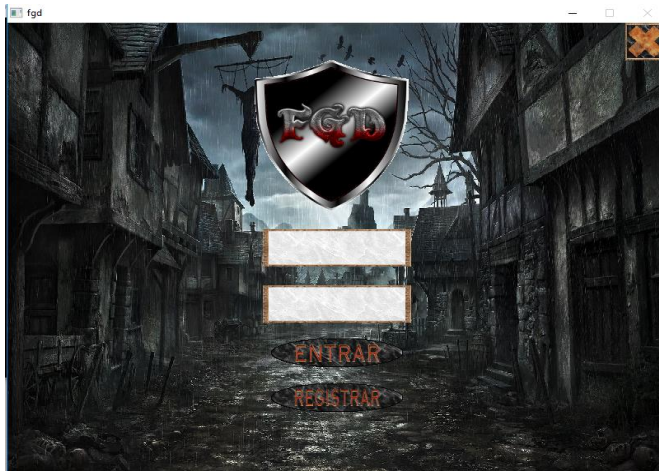
- Espai: ataca

Tecles especials

- I: obre el menú d'inventari
- Escape: obre el menú

Manual d'usuari

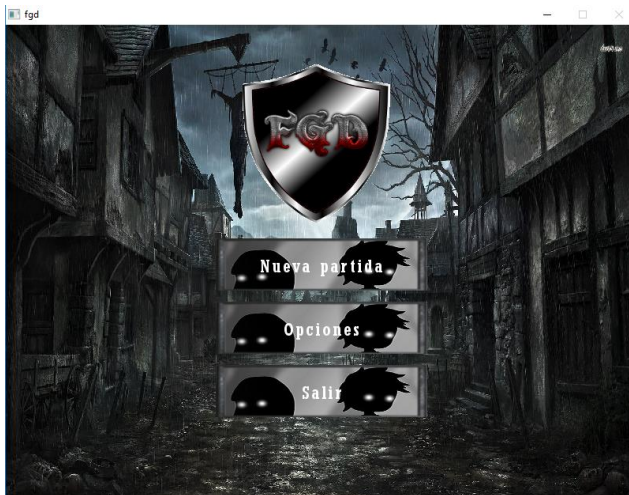
Login



Al executar l'aplicació s'obrirà el menú de login.

En cas de tenir un usuari ja registrat introdueix l'usuari i la contrasenya i fes clic a entrar.

En cas contrari teniu que clicar en registrar, introduir l'usuari i contrasenya que voleu i pressionar a terminar



Iniciar Partida

Per iniciar partida tens que estar en el menú principal de l'aplicació i fer clic a “NUEVA PARTIDA”

Moviment del personatge

Per moure el personatge per el mapa teniu que utilitzar les tecles de moviment: Fletxa amunt, Fletxa dreta, Fletxa abaix, Fletxa esquerra.

Atac a enemics

Per atacar als enemics teniu que utilitzar la tecla de atac: “espai” la vida del enemic canviarà depenent de la vida actual que tingui.

Desactivar la música

Per desactivar la música es pot fer des de dos llocs: el menú principal i des de el menú al fer clic a la tecla escape

-menú principal->opcions->música.

-dins de partida->tecla escape->opcions->música.

Guanyar partida

Per guanyar partida teniu que arribar a l'últim nivell de lloc i matar al boss final.

7. Valoració global del projecte

Com a valoració nostra que hem tingut del projecte nosaltres estem contents amb el treball que hem pogut realitzar, ens hem ficat de ple a un terreny totalment desconegut per nosaltres, unes tecnologies que no havíem utilitzat mai i a més molt difícils com es el llenguatge programació orientada a objectes C++, que amb això ens hem trobat amb molts problemes com son els punters però hem aconseguit portar el projecte endavant.

Tot això ho hem aconseguit ya que hem anat amb molta il·lusió i ganes a un projecte que no sabíem si ho podríem realitzar ya per no tenir res de coneixements en aquest sector, dit això i el que hem mencionat estem contents de haver pogut realitzar el primer desenvolupament d'un joc (d'aquí el nom de FGD 'first game develop'), ya que era el objectiu que teníem des de fa 2 anys el poder desenvolupar jocs, ens ho hem proposat i ho hem aconseguit.

La planificació i realització d'aquest treball de grup es una bona practica per poder aprendre i investigar sobre el teu treball, hem après a aprendre, hem après a treballar en equip que des de el nostre propi criteri es una practica que creiem que falta durant el curs el practicar en treballar en equip, hem tingut que investigar moltíssim per poder realitzar aquest treball, un treball que si no s'explica be tot el que te enrere no es pot entendre tota la dificultat que pot tindre, es pot veure fàcil quan no ho es, això es una cosa que ens hem trobat fins a acabar-ho i abans de explicar el que hem tingut que fer.

Després de desenvolupar en un llenguatge i àmbit tan complexa com es el d'un joc i el C++, gestionar tot per el nostre compte ya que hem a sigut tot de 0 sense motor de joc ni res, a més de anar a intentar a fer-ho el millor possible ya que volíem presentar un treball consistent que es pugui ver tota la funcionalitat i treball que te i tot el que es pot arribar a aprendre d'això, per aquest mateix motiu el nostre treball esta tot pujat a GitHub per a persones com nosaltres que hem tingut que buscar mil fonts d'informació i manuals només per trobar una cosa, porque tinguin facilitat per entendre els camins i procediments basics que te qualsevol joc internament.

Dit tot això ens ha a agradat molt realitzar aquest treball que hem disfrutat desenvolupant i hem trobat l'objectiu que volem seguir a partir d'aquí

8. Bibliografia

- Instal·lació d'allegro en Windows
https://www.youtube.com/watch?annotation_id=annotation_232398943&feature=iv&src_vid=1hvMA5fpxfs&v=xjQHQzhmOQ8
- Documentació api Allegro 4.0.1
<http://es.tldp.org/Allegro-es/web/online/allegro.html>
- Exemples UML
http://osl2.uca.es/wikijuegos/w/index.php?title=Temario/Un_ejemplo_de_la_creaci%C3%B3n_de_un_videojuego/An%C3%A1lisis
- Informació coneixements bàsics per la creació de jocs amb Allegro 5
<http://fixbyproximity.com/2d-game-development-course/>
- Col·lisions 2D:
<https://www.genbetadev.com/programacion-de-videojuegos/teoria-de-colisiones-2d-conceptos-basicos>
- Documentar c++
<https://stackoverflow.com/questions/1141228/javadoc-like-documentation-for-c>
- Tutorial Allegro 4
http://www.evc-cit.info/cit020/allegro_tutorial.pdf
- Text Generator
<https://cooltext.com/>
- Col·lisió circular amb quadre
<https://stackoverflow.com/questions/401847/circle-rectangle-collision-detection-intersection>
- Menú allegro
<https://www.youtube.com/watch?v=Vrjg8F4U22Y>
- Enemy simulant una IA:
<https://www.youtube.com/watch?v=wWb4FP9R0CQ>
- Sprites
<https://forum.chaos-project.com/index.php?topic=2695.0>
<http://www.mundo-maker.com/t5495-pack-de-mundo-maker-2012>

- Timers

https://wiki.allegro.cc/index.php?title=Using_Timers_in_Allegro_4