Game title: Escape the Hive

Author: Stroici Andrei

Game play

This is a single player game. To win this game you must pass 3 levels, with different difficulty levels, from easy to hard. You can go in eight directions (as you will see in the pictures from below where I draw the map of all levels in section Mechanics). Every level has a number of enemies that can move same as you, but their move is random. You have power to throw projectiles, fire balls that can kill enemies, but they cannot destroy walls. When an enemy is killed, they can change their position on the map or it is possible to kill two or more of them in the same time. If the main character touches his projectile or an enemy touch him, he dies, and the game is over. When this happens, the game is closed because, the main character is dead.

Plot

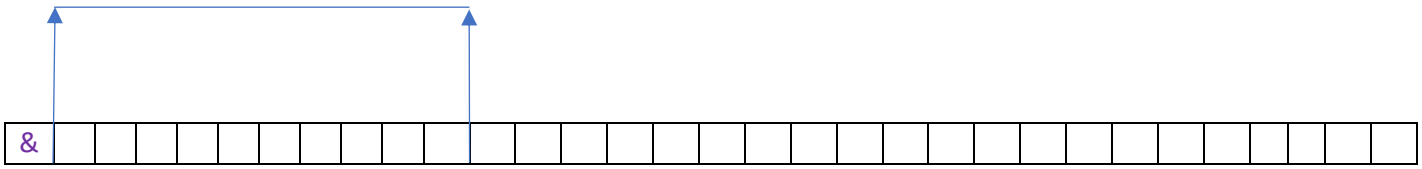
You are a human that tries to escape from an oppressive world ruled by robots. To do this you need to pass 3 zones guarded by robots. With every step you come closer to the exit of labyrinths, and your freedom, the number of bad robots is increasing. You can shoot fire balls, and destroy your enemy, but there is a problem. The robots can teleport all over the map when they detect that one of them is killed. Also, they have a problem at construction and some of them don't have good batteries and energy reserves, so they need a master that supplies them energy. If you manage to kill the master, automatically, the robots that have energy from master die and disappear.

Characters

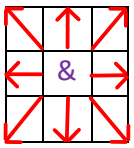
- Zoltan is the main character of the game.
- Enemies: they are the robots that you will see on the map and you will try to destroy or escape.

Mechanics

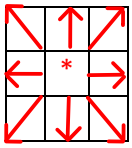
Projectile range



The directions in which player can move



The directions in which enemy can move



- Level 1

-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
&																															-
-																		-	-	-											-
-																		-	-	-											-
-						-	-	*										-	-	-				-	-	-	-				-
-						-	-																	-	-	-	-				-
-																													-	-	-
-									-	-											*								-	-	-
-									-	-																					-
-												-	-																		-
-												-	-																		-
-																								-	-	-					-
-			-	-	-											-	-							-	-	-					-
-			-	-	-											-	-														-
-			-	-	-				*										-	-								-	-		-
-										-	-								-	-								-	-		-
-										-	-																			*	^
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

- Level 2

-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
&																-		-											-	-	-
-	-		-												-														-	-	-
-				-																										*	-
-	-		-	*					-	-								*										-	-		-
-									-	-															-	-		-	-		-
-														-	-										-	-					-
-														-	-					-	-										-
-																				-	-				-	-					-
-		-					-	-																	-	-					-
-			-				-	-								-	-														-
-				-								*				-	-														-
-			-				-	-																							-
-		-					-	-												-	-	-	-							*	-
-			-	-										-	-					-								-	-	-	-
-			-	-										-	-	-	-								-	-		-	-	-	-
-	*																								-	-					^
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

➢ Level 3

-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
&																			-		-		-		-		-		-	-
-	-		-		-		-		-		-		-		-		-							-						-
-																									-						-
-	-		-		-		-		-		-		-		-		-		-		-		-		-	-	-		-	-
-																		*												*	-
-				*																											-
-	-		-			-				-				-	-	-	-	-	-	-			-			-			-		-
-		-		-													-														-
-	-				-						*						-														-
-	-			-		-			-	-				-			-			-			-			-		-		-	-
-									-																						-
-									-															-	-		-	-	-	-		-
-	-		-		-		-		-		-		-		-		-		-		-		-	-		-	-	-	-		-
-																								-	-					-		-
-	-		-		-		-		-		-		-		-	*	-		-		-		-	-	-	-	-	*	-	-	-
-	*																							-	-	-	-	-				^
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Legend

- Wall

* Enemy

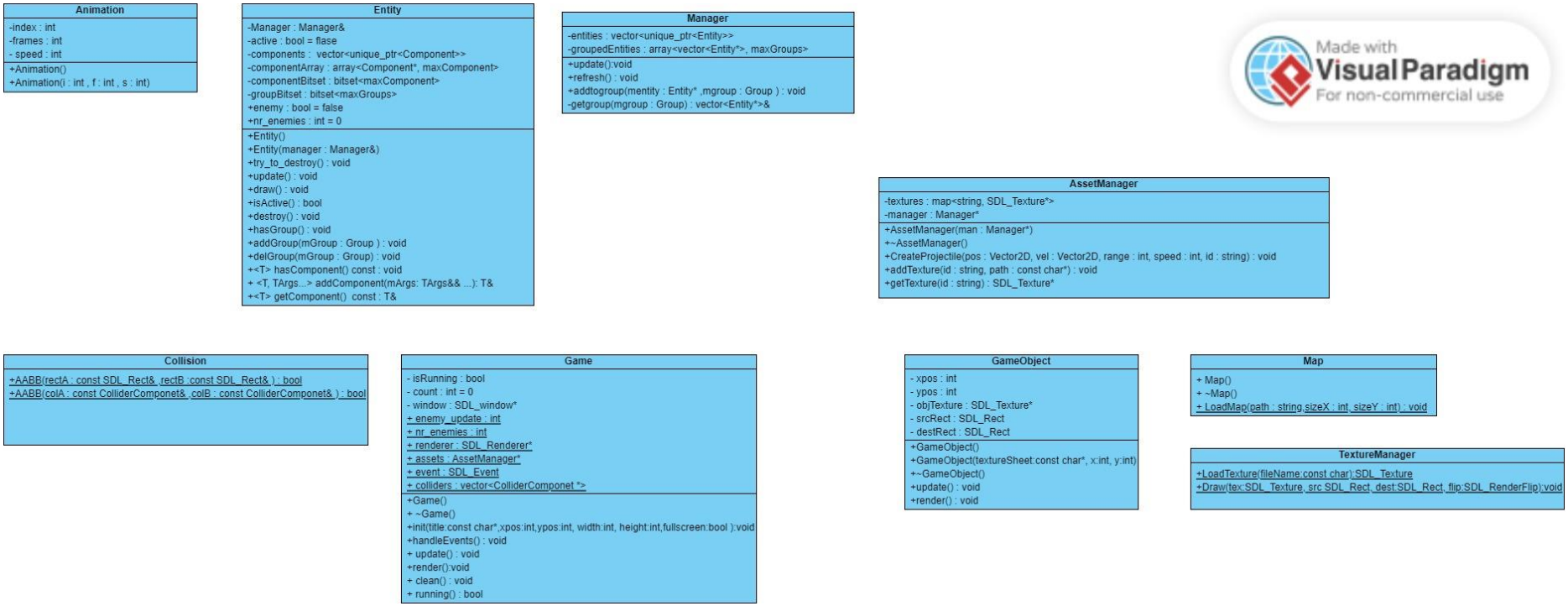
& Start position

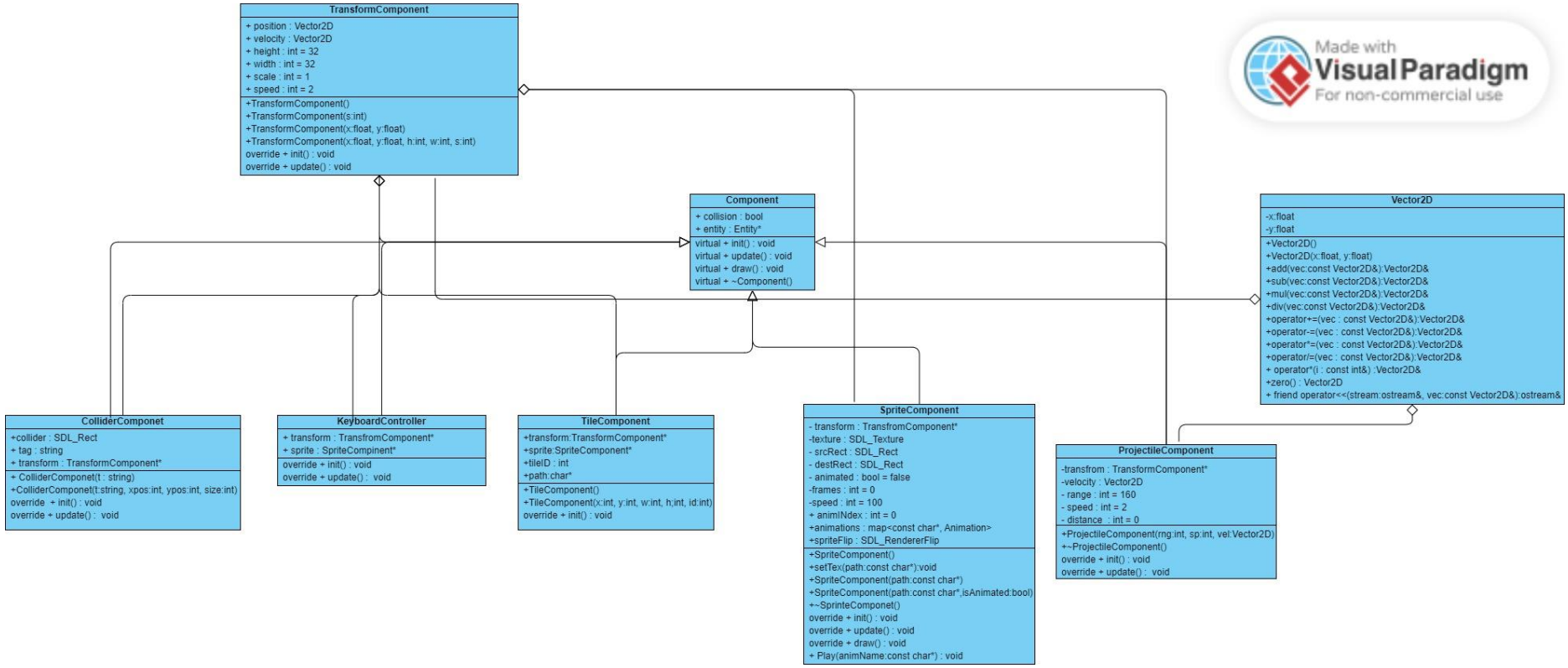
^ End position

Where is nothing is grass

zone where the player can walk

Classes and Components




TransformComponent
+ position : Vector2D
+ velocity : Vector2D
+ height : int = 32
+ width : int = 32
+ scale : int = 1
+ speed : int = 2
+TransformComponent()
+TransformComponent(s:int)
+TransformComponent(x:float, y:float)
+TransformComponent(x:float, y:float, h:int, w:int, s:int)
override + init() : void
override + update() : void
Component
+ collision : bool
+ entity : Entity*
virtual + init() : void
virtual + update() : void
virtual + draw() : void
virtual + ~Component()
Vector2D
-x:float
-y:float
+Vector2D()
+Vector2D(x:float, y:float)
+add(vec:const Vector2D&):Vector2D&
+sub(vec:const Vector2D&):Vector2D&
+mul(vec:const Vector2D&):Vector2D&
+div(vec:const Vector2D&):Vector2D&
+operator+=(vec : const Vector2D&):Vector2D&
+operator-=(vec : const Vector2D&):Vector2D&
+operator*=(vec : const Vector2D&):Vector2D&
+operator/=(vec : const Vector2D&):Vector2D&
+ operator*(i : const int&) :Vector2D&
+zero() : Vector2D
+ friend operator<<(stream:ostream&, vec:const Vector2D&):ostream&
ColliderComponet
+collider : SDL_Rect
+ tag : string
+ transform : TransformComponent*
+ ColliderComponet(t : string)
+ColliderComponet(t:string, xpos:int, ypos:int, size:int)
override + init() : void
override + update() : void
KeyboardController
+ transform : TransfromComponent*
+ sprite : SpriteCompinent*
override + init() : void
override + update() : void
TileComponent
+transform:TransformComponent*
+sprite:SpriteComponent*
+tileID : int
+path:char*
+TileComponent()
+TileComponent(x:int, y:int, w:int, h:int, id:int)
override + init() : void
SpriteComponent
- transform : TransfromComponent*
-texture : SDL_Texture
- srcRect : SDL_Rect
- destRect : SDL_Rect
- animated : bool = false
-frames : int = 0
-speed : int = 100
+ animINdex : int = 0
+animations : map<const char*, Animation>
+spriteFlip : SDL_RendererFlip
+SpriteComponent()
+setTex(path:const char*):void
+SpriteComponent(path:const char*)
+SpriteComponent(path:const char*,isAnimated:bool)
+~SprinteComponet()
override + init() : void
override + update() : void
override + draw() : void
+ Play(animName:const char*) : void
ProjectileComponent
-transfrom : TransformComponent*
-velocity : Vector2D
- range : int = 160
- speed : int = 2
- distance : int = 0
+ProjectileComponent(rng:int, sp:int, vel:Vector2D)
+~ProjectileComponent()
override + init() : void
override + update() : void
Made with
VisualParadigm
For non-commercial use

Game States

The game has 3 levels. To go to next level, you need to go to a specific place on the map. While are you trying to achieve you should be carefully not to touch your projectiles or any enemy. If this happens you will die, and the game will close itself. To restart the game, you need to launch the game again. When you will win you will see a message in console, with "You win". You can go in all neighbor cells by using UP, DOWN, LEFT, RIGHT or 'w', 's', 'a', 'd', keys. When you will achieve the exit point you will go automatically to next levels or you will win the game.

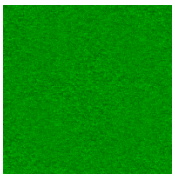
Assets and Resources

1. Player asset

2. Enemy asset

3. Projectile asset

4. Grass asset



5. Wall asset

To create the game I used a tutorial found on the YouTube on this link: https://www.youtube.com/watch?v=QQzAHcojEKg&list=PLhfAbcv9cehhkG7ZQK0nfIGJC_C-wSLrx&pp=iAQB.

This is the start point I used to create my game. During implementation I was forced to add some new parts in the code. For example I needed to create the levels, to bind them and to make transitions to the next level.

Technical Details

For this game I used the C++ language, and the orientation object programing paradigm. I tried to create a system of components (class Entity see in the UML diagram at Classes and Components). This class represent a general form for an object present in the game, like player, wall or enemy. In this class I check if the entity is active, that is a member that show me if the entity should be destroyed or not. I use also some containers to keep information about the entity, like components (position, animation, sprite), but also to have an easier access to them I keep them in groups. Manager class is responsible to watch after all entities.

Component class is an interface for all proprieties for an entity. Using the polymorphism propriety, I can override all operation of this class and create proprieties for entity that they modify as they need.

The position on the map is implemented with class TransformComponent. By using class Vector2D I represent a place on the map, but also the direction that entity moves, position and respectively velocity members. Also, here I keep information about the dimension of the object and the scale. For this game I choose to keep the cell's dimensions of 32 x 32.

Using SpriteComponent I manage the sprite and with class named Animations, the animations of the player and enemies. In this class I use a bool variable to know if the entity is animated and also, give informations about the speed of the frames, the numbers of frames per second and the period between frames.

With tile component I create and modify the elements on the map. Every cell in the map has a texture and an ID, that tells me what is in a position, specified by the attribute of type TransformComponet.

ProjectileComponent is the class that implement the projectile. It gives information about the position of the projectile on the map (attribute named transform), the way it moves(the attribute named velocity – this tell me how to update the position of the projectile), but also the speed, range and the distance that projectile traveled.

Class named KeyboardControler is used to read the player input, and to update the velocity component of the player entity. Here I let the player to have the chance to choose what keys is used : UP, DOWN, LEFT, RIGHT keys or the w, s, a, d keys.

Using the class ColliderComponet I create an area that is collidable and see what entity has a collision. With Collison class I can detect the collision and decide what to do : not to move if the enemy or player hit a wall, to kill the enemy if a projectile hit him, or to end the game if the player hit an enemy or a projectile.

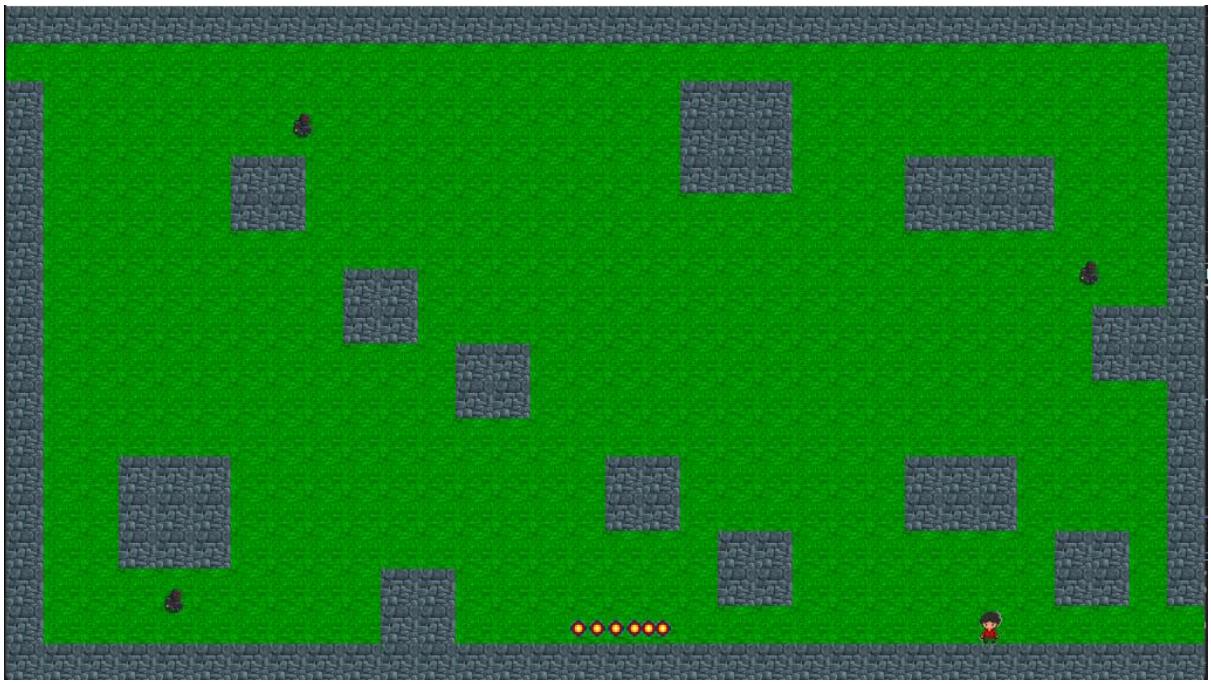
Map class is used to manage the map. In this class I have a function that read from a file, found in the asset, every map for levels, anytime I detect that the player won a level.

TextureManager class is used to load the textures for every entity and object is displayed.

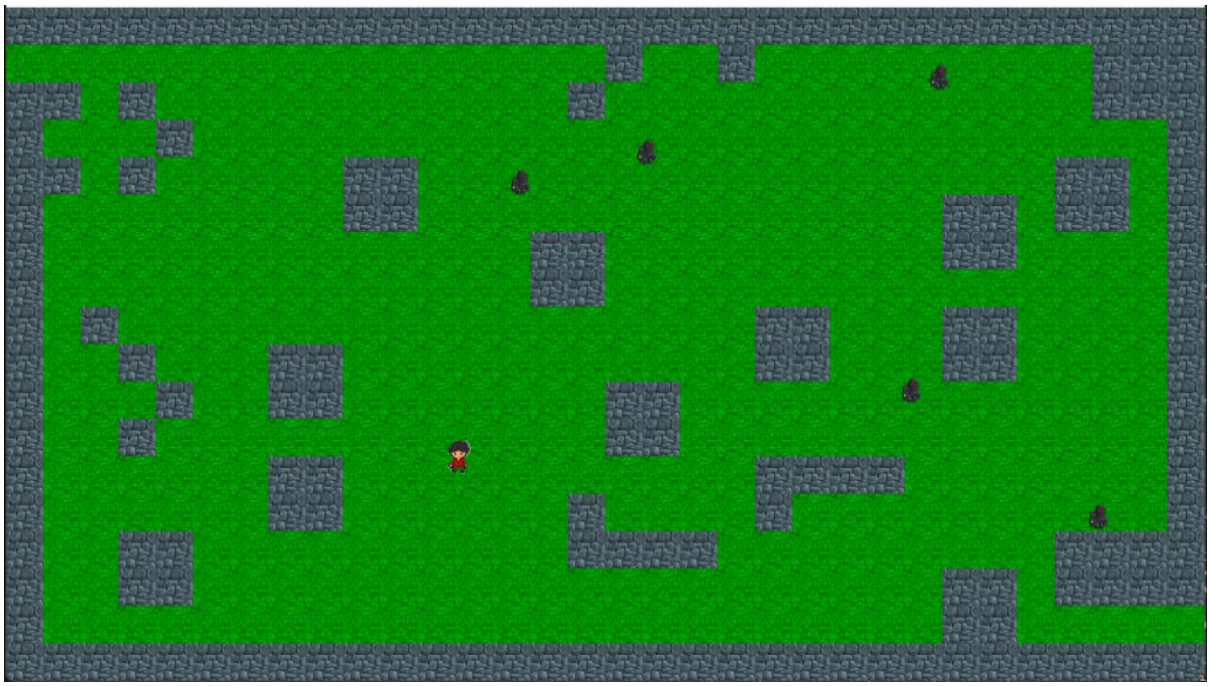
Game class is responsible for the running of the game. In this class I initialize the game : I tell to open a window, the size of the window, and load the map, and display the player. After I use this class to update every component used in the game, handle the key events, working together with the class KeyboardController. Also, this class is responsible to render all components.

Images from the game

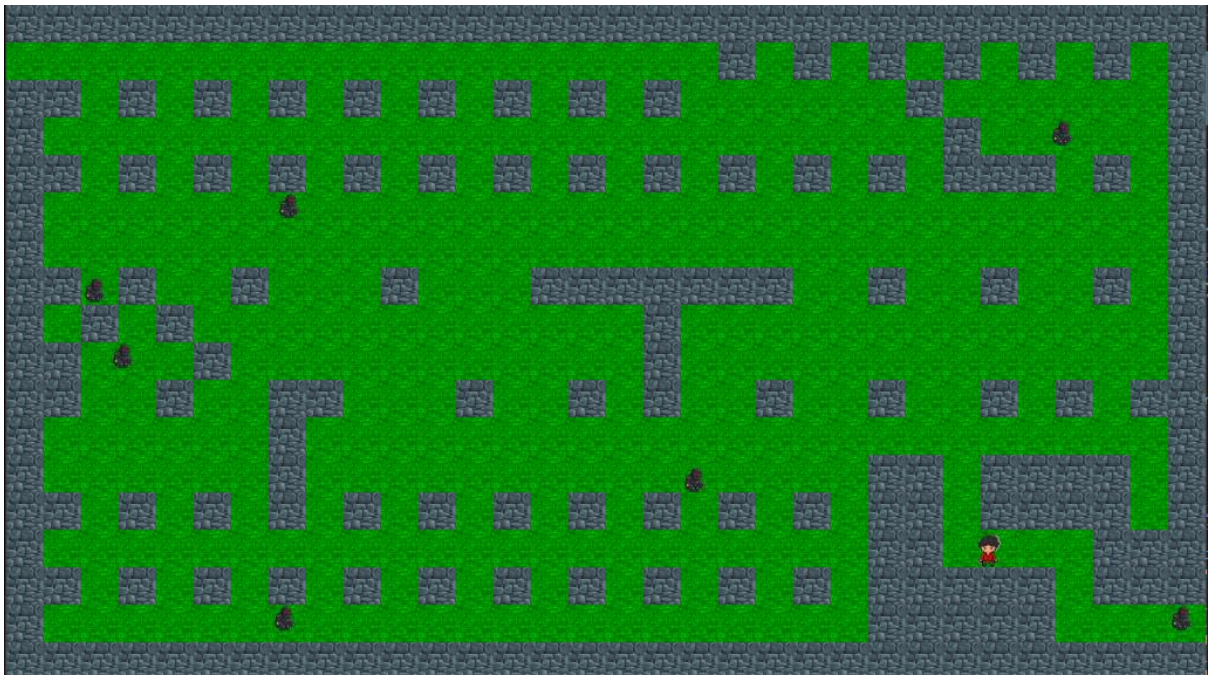
Level 1



Level 2



Level 3



Notes and Considerations

I already know that in the code exits a bug, but I didn't manage to solve it. This problem is when the player kill an enemy. Sometimes it kills more than one enemy or move them on the map. I tried to cover it using a story. I plan to try during summer holiday to review the code and do some optimization and to fix the bug, and add an interface to the game, and more levels.