CMP 105 Coursework Report:  
 Han Vellsing  
  
Radu Groza  
2002046  
Git Username: AxeIG  
  
  
Introduction  
Han Vellsing is a platformer in a gothic setting similar to games like CastleVania or Blasphemous.  
  
Quick summary about the game  
Mention how you handled animations  
Idea behind the game  
game type  
unique mechanics  
collision detection  
audio  
  
Controls  
The game uses simple controls used in most platformers. The Player can move by pressing the WAD keys and attack by pressing the Enter button.  
  
Game screen  
The game uses one main level screen which consists of two layers, one of them being a stationary city and the second one being the clouds/scenery which loops in the background.

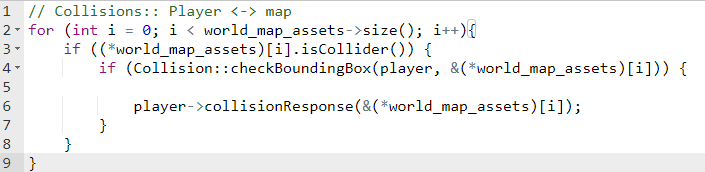


The background looping is done by doubling the photo by itself and then using a rectangle that has the size of the original photo to go through it. In the level update, using the updateTextureCoordinates function of the background. The rectangle will move one pixel to the right through the original photo every frame and once the rectangle reaches the  
other half of the image the rectangle will be redirected to the very beginning of the image which represents an identical image, then it starts moving again, giving the illusion of looping. Instead of creating a whole background manager as in the reference [1] I decided to use implement it in the background class since I have only one background.  
  
The rest of the game uses simple transparent text messages as game screens, such as a “you died” except for the Main and Tutorial screen which showcase an entire PNG image. The different game screens also include various buttons that change the game state if pressed.

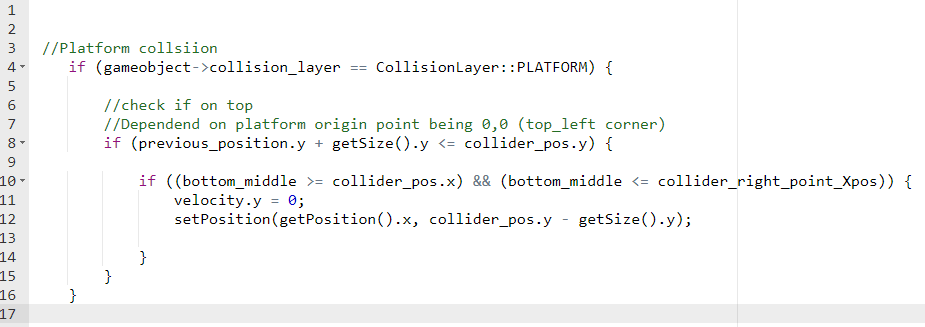


Input  
The player has access to both vertical and horizontal movement using keyboard buttons.  
By pressing A or D the player will move horizontally and when pressing W the player will jump, based on how long he holds the button.  
The player can also attack enemies by pressing the Enter key whether he is on ground or not.  
Go through the player Handle Input

Sprite work and Animations  
  
For animation handling the characters are using two functions, handleAnimation and AssignAnimation. The first one decides what animation is most suitable for the character to use based on his game state machine and then calls the AssignAnimation function to assign said animations. All derived character classes (e.g. imp and player) have to override their own handleAnimation to fit their own behaviours but the AssignAnimation remains the same for all character objects.  
After I got my game assets online, I started encountering some difficulties with them. I had to redo most of the sprite sheets myself and the different character animations were not consistent which required me to manually change the size of the player object after I assign him a new animation.  
In order to fix this I created a separate game object called sprite that manages the sprite’s position and size in the AssignAnimation function relative to the that of the character’s centre position.   
A small advantage of using a separate game object to manage animations is that I can freely resize the character and play with him without fear that the sprite itself will stretch in order to fill the character rectangle.  
Another advantage is that for some specific collision response events I could use the sprite’s collision box instead and use the character’s for different purposes  
  
  
Collision Detection and Response  
  
For Collision detection and response I have implemented collisions layer to each game object using an enum class.  
The map tiles consists of walls and platforms that are detected only by the player.

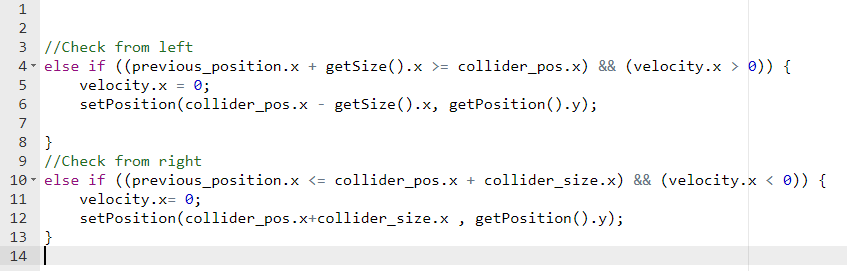


If the player collides with a platform it will only perform a top check. If the middle bottom of the player was above the tile in the previous frame, then the collision will occur, the player will be placed on top of the platform and his y velocity will reset

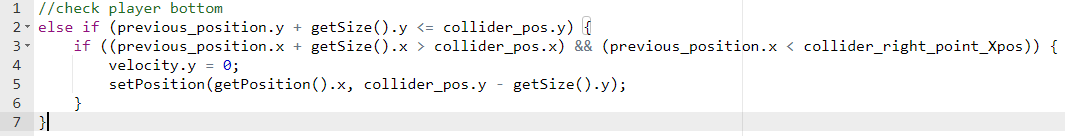
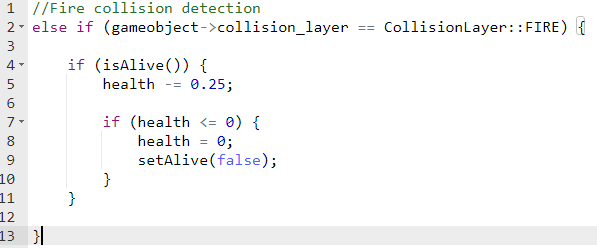
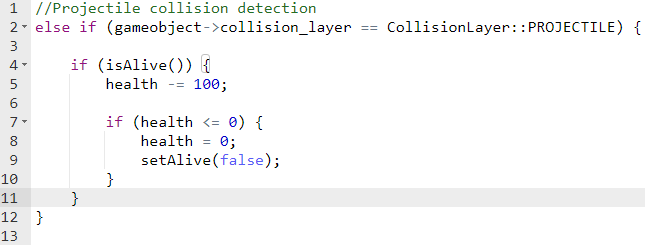


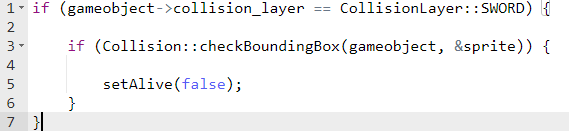
The collision box for the tiles is also slightly smaller so that it is slightly harder for the player reach them with a long jump. This results in small collider gaps between the platform tiles but the character collision width is way greater than that, making the chance of the character falling through it impossible.

   
  
If the player is colliding with a wall the collision check will be from all sides. The player’s collision detection with the wall and platform’s top occurs the same. For the wall’s sides, the player check’s if he’s previous upmost right and left collide with the wall, then he is pushed onto one of the wall’s sides based on the collision side and the x velocity will be reset



The player’s collision detection with the bottom of the wall works in a similar fashion, if the player’s previous top collides with the wall’s bottom, the player gets pushed and his y velocity resets, making him fall.

  
  
If the player is colliding with an enemy ( the imp) he will check for his two possible attacks.  
If the player collides with the imp projectile he will simply loose an x amount of health on hit,   
if the player collides with the tornado fire attack, he will loose a small amount of health for each frame in which he is getting hit by the tornado.  
  
  
   
  
From the imp’s perspective, he can only collide with the player sword.  
The sword is a player game object that just represents different collision box from that of the player which only updates when the player performs an attack, making the sword active.  
If the sword is active and the imp’s sprite interacts with it then the imp will die.

  
  
Audio  
Talk about how “I” used the audio manager  
  
  
  
   
Game Logic and Unique Mechanics  
  
  
  
  
  
  
  
  
Conclusion  
Mention the new object oriented programming principles that I have learned, the class managers, working with sprites and sprite sheets.  
  
  
  
  
References

[1]Background looping :https://en.sfml-dev.org/forums/index.php?topic=17370.0  
  
Bobby  
sounds from free sounds + some made by myself;