CS-A1121 Y2 Project Plan

Platformer

Oskar Björkgren 705868,

Kone- ja Rakennustekniikka, 3rd year

20.02.2022

General description and difficulty level

The goal for this project is to make a fully functional platforming video game. I will be striving for reaching the hard requirements. The requirements are as follow:

Medium

- Graphical user interface
- Graphics can consist of circles and squares
- Functioning collision detection. The program has to recognize when two or more game objects collide. For example, the program has to be able to tell if the player character runs into an object, and prevent the character from moving through the object.
- The game has to have clear conditions for winning and losing.
- The game has to have at least one premade level
- Pay attention to extensibility (for example, addition and creation of new levels)
- Unittests for at least part of the program

Hard

- Multiple levels
- A score counter or a timer, and a scoreboard of the top scores
- Keyboard shortcuts
- The possibility to save scores, at least to the scoreboard

So, with these requirements in mind, I will start to develop a simple platformer, where a character navigates through a map/level to reach an end destination as the win-condition.

The project also requires some sort of unique aspect that makes it different from the others of the same topic. I have a few options in mind; **adding enemies to the game**, add animations (at least to the player character), **add some sort of combat against the enemies** (example: basic sword attack/jumping on them ala. Mario), add some sort of hook-shot type weapon to add an element to the platforming.

Use case description and draft of the user interface

A platformer doesn't really need a traditional user interface except for maybe a start/pause menu that takes mouse or arrow key input. The game communicates with the user by moving the player character using keyboard keys.

Program's structure plan

For my initial plan the game will be initialized and run through the main game class. Things like collision detection will also probably be run through functions inside the game class. As the game is in real-time the game class will also have a time function that keeps the game constantly updating.

Player and enemy types will inherit from an abstract character class. The character class will handle when player/enemies take damage, but movement will be handles in the respective classes because they will have to be different. Player must be able to take input from keyboard and translate it into movement on screen.

The Map class will handle building the map, placing the obstacles, platforms, and enemies in the right places.

Gui class will be used to set up sprites for all the things that need them. Scaling sprites and updating them can be handled thought here.

Saved scores will be a file that is read into game class and is saved into when completing level/map. The ScoreCounter data is written into the SaveScore file.

See end of document for UML.

Data structures

Normal data structures available in python should be enough. Object lists and arrays should suffice for my needs.

Files and file formats

The files my program needs to handle are at least PNG images files for the different textures I need for my character, background, and platform sprites. These are easily read with PyQt functions. The program also needs to read and write into a text file for the saved high scores. This is easily done with normal python methods.

Algorithms

The most “complicated” algorithms I can see myself needing in the project will be for collision detection and enemy AI/movement. Which basically boils down to calculating the distance between two points/objects. Enemies might have different AI like just walk in straight line until it can’t anymore or find player position and walk towards it. Also, the Wikipedia page on collision detection will be helpful.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Testing plan

A few test cases I have in mind is checking that characters, platforms and obstacles are created correctly and spawned in the right places.

Checking that keyboard input is valid and does the correct thing on screen, example move character some amount to the left/right.

Damage testing for enemies and players if there is collision.

Libraries and other tools

For the time being I plan on only using PyQt.

Schedule

Week 1

Get basic framework done, game loop and movable character.

Week 2

Start working on building the game map, player can interact with the world. Write some tests for this

Week 3 (checkpoint 1)

Have a working world which the player can move through by jumping and falling down pitfalls to end game. Finished collision detection. Continue writing tests.

Week 4

Work on enemies and their movement in the world

Week 5

Player – enemy interactions.

Week 6 (checkpoint 2)

Start working on hard requirement features, add levels, score counter, etc…

Week 7

Finish hard requirements features.

Week 8 and Week 9 (deadline/final version)

Fine tuning gameplay, minor bug fixes, documentation and writing final test cases

Schedule comment

I plan on using about 10 hours each week (broken down 2h each working day) on the project. The weeks will probably not be exactly like this, but this is just a general outline for time management.

Literature references and links

PyQt5 reference guide:

https://www.riverbankcomputing.com/static/Docs/PyQt5/introduction.html#pyqt5-components

Collision detection:

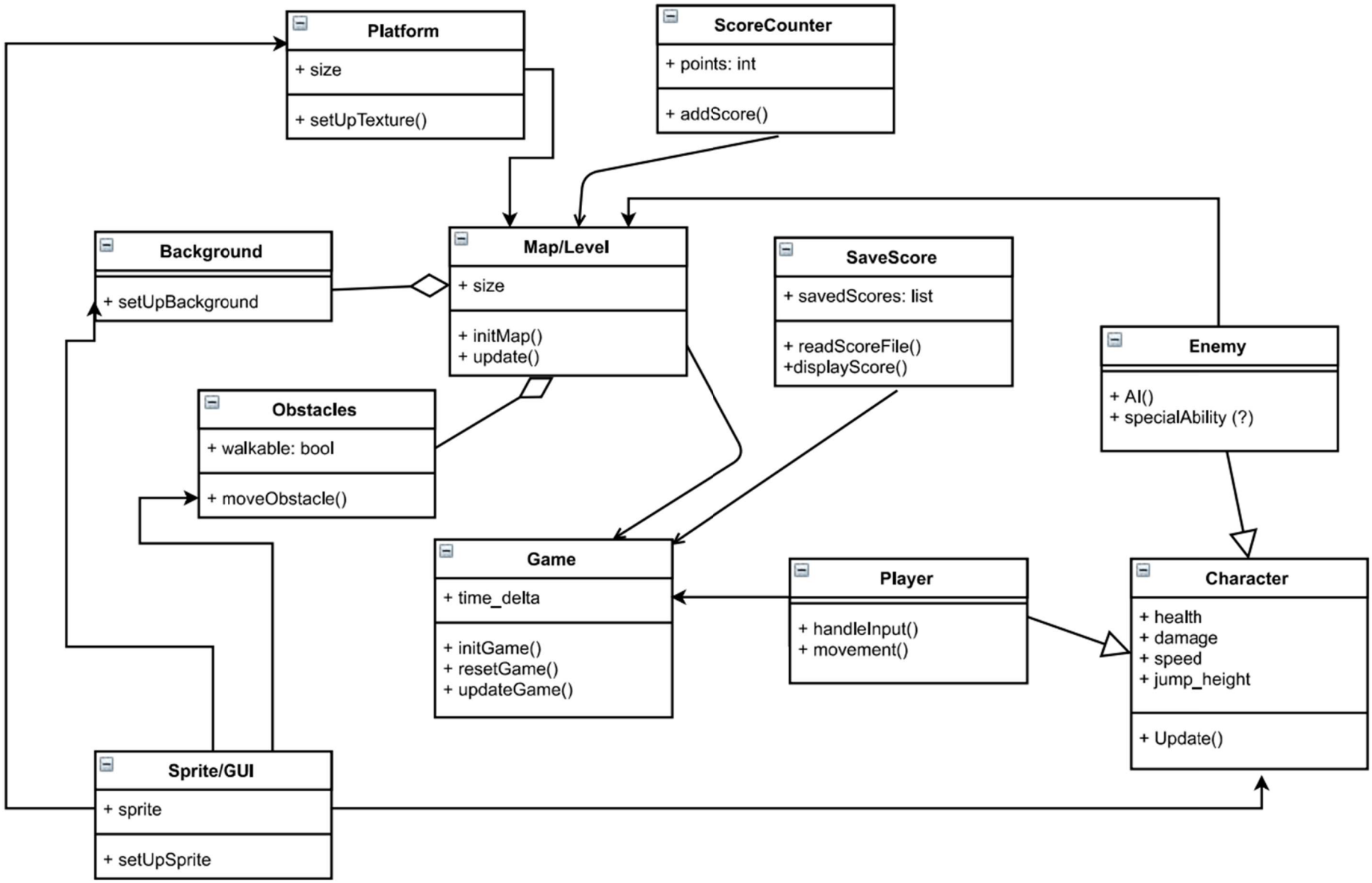
https://en.wikipedia.org/wiki/Collision_detection#Video_games

Qt Based Games:

https://wiki.qt.io/Qt_Based_Games

Creating GUI app youtube series:

https://www.youtube.com/watch?v=9iZLDnW_vwU


Platform
+ size
+ setUpTexture()
ScoreCounter
+ points: int
+ addScore()
Background
+ setUpBackground
Map/Level
+ size
+ initMap()
+ update()
SaveScore
+ savedScores: list
+ readScoreFile()
+displayScore()
Enemy
+ AI()
+ specialAbility (?)
Obstacles
+ walkable: bool
+ moveObstacle()
Game
+ time_delta
+ initGame()
+ resetGame()
+ updateGame()
Player
+ handleInput()
+ movement()
Character
+ health
+ damage
+ speed
+ jump_height
+ Update()
Sprite/GUI
+ sprite
+ setUpSprite