CS-A1121 Y2 Project

Platformer **Janky Tower Escapade**

Oskar Björkgren 705868

Kone- ja rakennustekniikka 05/05/2022

General description

The game is a classic platformer where you can move around a player character, jump on platforms, and avoid traps. In the game there are two types of traps: fixed spikes and moving spiky balls. The goal of the game is to climb the tower while avoiding the traps and to collect coins placed through out the map. Upon death you are transported back to the beginning of the map and must try again. You can't exactly loose the game but when you die all coins return and you must try again. There is a score counter that calculates that gives you a score based on how many times you died, how many coins you made it with to the end and the time it took to escape the tower. You can also pause the game, which gives you a menu for scoreboard and exiting the game. The scoreboard button opens a message box from which you can see all the scores, but the exit button just prints a to the terminal that you have to exit manually from the window.

The general structure of the game is different from the original plan. I scraped a lot of things like killable enemies and replaced them with unkillable moving obstacles because they were a lot easier to implement. Most of the game logic is handled through the Game Loop class. A lot of the planned classes where also not needed because of PYQT5s functionalities and ready-made classes/widgets.

The requirements:

Normal

- *Graphical user interface*

  Basic 2D pixel graphics

- *Graphics can consist of circles and squares*

  Added a little bit of extra character to the game with the traps and player character nothing major

- *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.*

  There is a collision detection system, but it's a bit it leaves a bit for improvement. There are a few bugs that make you fall a bit into the ground texture, but it effects the gameplay only visually. There are also instances of when you can jump a bit into platforms, but it works as a ledge grab and makes the platforming a bit easier, so I can see it as a feature and not a bug.

When picking up coins and colliding with killing obstacles the collision detection works well.

- *The game has to have clear conditions for winning and losing.*

  Climb the tower to win and die when you are hit by the obstacles.

- *The game has to have at least one premade level*

  The game has currently two ready made maps, but more could easily be added

- *Pay attention to extensibility (for example, addition and creation of new levels)*

  Addition of new levels is pretty easy. You just have to make a build a new map according from the tiles that are in the game (make a new map settings file) and add it into the map list in the game settings file. New tile types and different obstacles are easily added to the map if you want.

- *Unittests for at least part of the program*

  This is very lacking I have a few.

  Hard

- *Multiple levels*

  The game supports more levels but more are easily added.

- *A score counter or a timer, and a scoreboard of the top scores*

  Exists, but kind of bare bones.

- *The possibility to save scores, at least to the scoreboard*

  Scores are saved automatically to the scores text file upon completion of the game.

  I think I have the normal requirements down and where they lack (a bit finicky collision detection and unittests) the hard requirements make up the difference. So normal difficulty.

  Instructions for the user

  The game can be run using normal python compiling. You can move the character left with the A button or right with D. Spacebar to jump. ESC to pause the game and to access scoreboard.

External libraries

Only PYQT5 has been used.

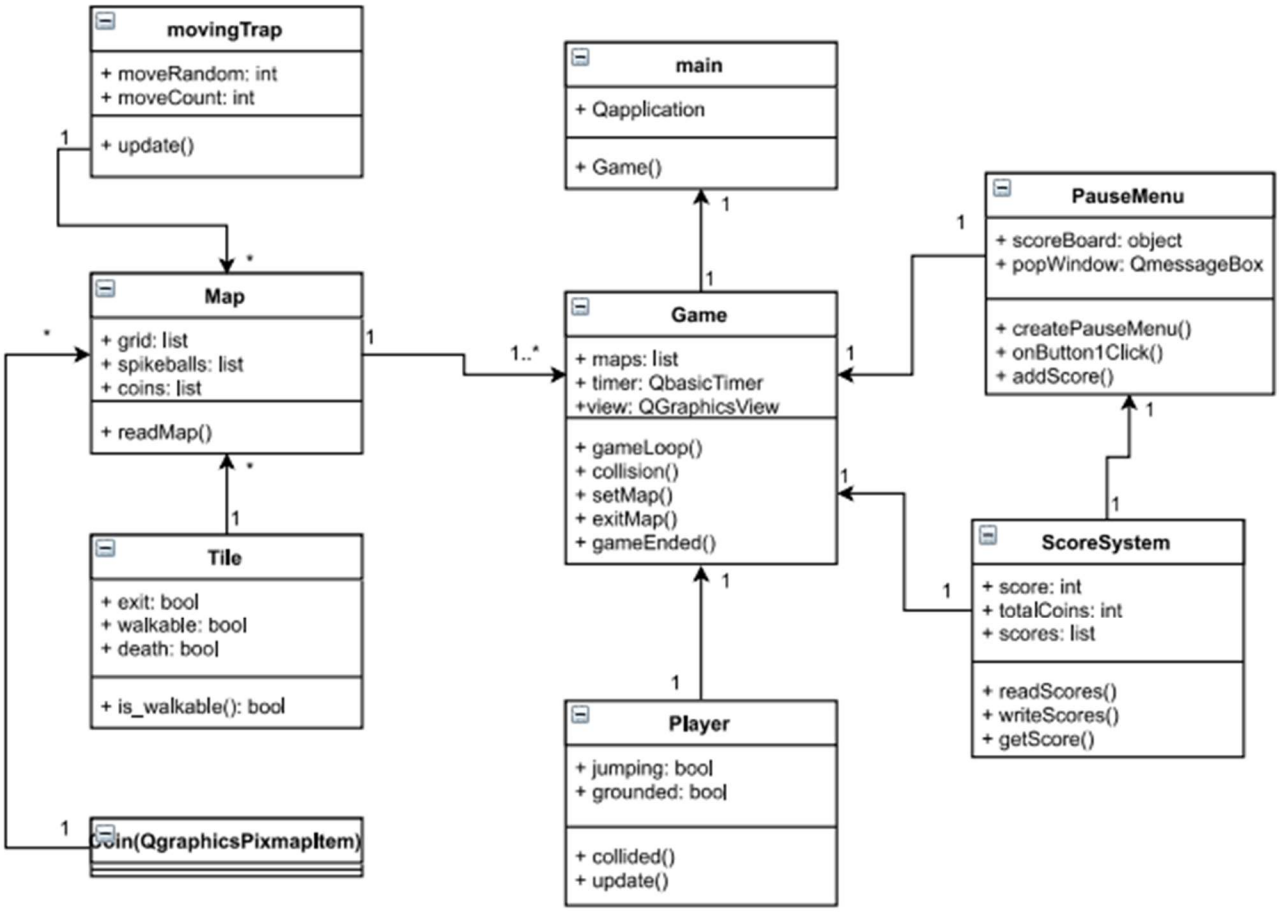
Structure of the program

The main game is started from the main file which creates the QApplication app which leads into the Game class through with the which handles most of the game logic. The Game class is a QGraphicsScene into which it is easy to draw all the wanted graphics items. The class Map doesn't inherit anything but reads a map structure from a map settings file which is then built in the Game class method setMap(). The method setMap() is very important for the extendibility of the game as it allows the game to update the current level on the QGraphicsScene into a new one. The Player, movingTrap, Tile and Coin classes all inherit the PyQT5 class QGraphicsPixmapItems because of the combability with the Game class QGraphicsScene.

One very important part of the Game class is the timer parameter, which is just a QBasicTimer, but it allows the game to update in real-time. Game class also creates a scoreboard from the scoreSystem class and a pauseMenu through the pauseMenu class. The scoreSystem doesn't have any parent class. It reads the scores from a text file, writes the score of completed game and keeps track of the currents score. The pauseMenu class however inherits the Qwidget class QToolbar. It has QMessageBox as a parameter which allows for a simple pop-up window to be created and it also creates two buttons from the QAction class, as it is a very simple way to get interactive parameters. The pauseMenu class also gets the scoreSystem class parameter created in the Game class so it can display the scores in the QMessageBox.

Based on the timer in the Game class a timerEvent method updates which runs the gameLoop() method that basically keeps all of the game running. It runs the update methods in all the moving part of the game (player and movingTrap). Runs the collision detection method and checks for if the game is paused. The collision function in the game class handles the collision between player and both coins and obstacles, but the collision between player and tiles mostly handled in the player class as it's a bit more complex than the other instances of collision.

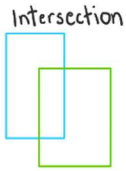
The UML below explains the different relations between classes and shows central functions of the classes.



Algorithms

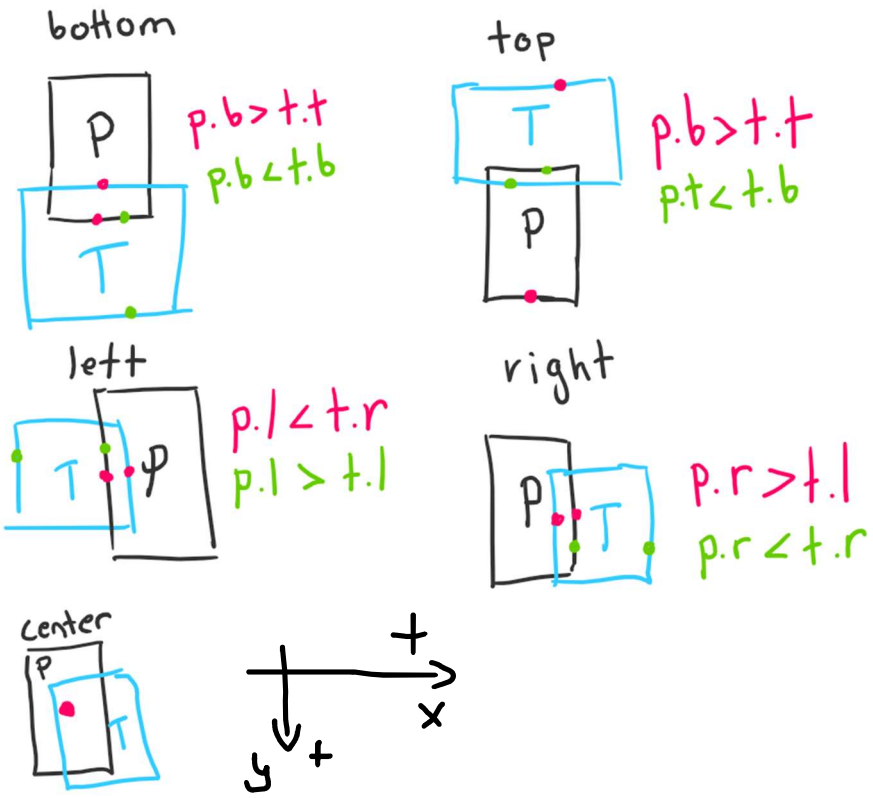
Collision detection

The most complex algorithm and maybe the most central one is the one for collision detection. There are two versions of the collision detection at work, a very basic one that just checks any intersection between items, and one that checks from which direction the collision is happening. I didn't program the first way which is gotten from a class method, but it checks if two items have any point inside each other based on position, width, and height. My collision detection work similarly and uses this function in the first part.



The player – tile collision work with 5 different checks which I have tried to illustrate bellow, the red point in center example is the position coordinates on the player characters middle.

P = player; T/(t before dot) = tile; t = top; b = bottom; l = left; r = right



So, for each case we have two checks and if both are true, we get true for that case. In the top and bottom case, we compare the y coordinates of the two items and in the left/right case we compare x coordinates. So based on which are true the player position is transported to old position before collision.

Jumping

The jumping algorithm is a very basic one, for each update we add to a check and when that check is over a certain number the jumping is turned off.

A better jumping algorithm would have been based on how long you press the jump button. The button press would height based on how long you press it which would allow for more controlled jumping movement. A check would have to be added so you can't jump into infinity.

Moving obstacle movement

Work similarly to the jumping algorithm and is based on a comparison check between movement done in one direction and a randomly generated number. When the movement is higher in one direction than the randomly generated number the obstacle turns around.

Data structures

Almost all relevant data structures are mutable lists. The map is built from a grid list of tiles, list of obstacles and coins. For getting the keyboard input I used a set as it worked very well with the keyPressEvent method. I didn't really see a need to use any other data structures.

Files

The program handles only text files and PNGs. The scoring is read and written into a text file in the format of

```
code = self.get_random_letter_code()
f.write(str(len(self.scores)+1) + code + ": " + str(self.getScore()) + '\n')
```

The different levels are created by map setting text files which are based on numbers that correspond certain tile types.

```
You, 2 hours ago | 1 author (You)
1  1, 1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1, 1, 1 ,1 ,1 ,1 ,1 ,1
2  1, 0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0, 0, 0, 0 ,0 ,0 ,0 ,6
3  1, 3 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0, 0, 0, 0, 0, 0 ,0 ,6
4  1, 0 ,0 ,0 ,0 ,0 ,0 ,1 ,1 ,1 ,1 ,1 ,1 ,1, 1, 1, 1, 1, 1, 1, 1
```

0: Background wall tile, 1: Platform tile 2: Spike trap

3: Coin, 4: Spike ball (moving obstacle)

5: Door to complete game, 6: Tile for moving to new level

Testing

The testing didn't go at all according to plan. The plan to was to write unittests, parallel to implementation of functions. But I am more accustomed to testing manually and using debugging function in visual studio code. I tested the function manually always when implementing them.

I have two unittests currently one for testing that the correct tile has been created and one for checking that Map class is created and one that catches if a map settings file is a file that doesn't work. I had more but I had a hard time getting them to work so I deleted them. The unittesting fell behind until the last week when motivation to write them was very low.

The known shortcomings and flaws in the program

The most obvious short coming is the lack of unittests. I should have implemented them when writing the code instead of trying to retrofit them to work with already written code.

The collision detection could also work a bit better there are some bugs that make the player character fall a bit into the floor and jumping into single tiled platform causes the player character to go into it a bit. But this bug works as a feature, essentially a wall grab, which makes platforming easier. This is easily the feature I spent the most time on, probably closer to 30 hours. Reading how QPixmapItems interact with each other, reworking it several times only to run into the same issues I had before.

The player movement is also a bit clunky, I explained earlier how I would have change jump movement to a more fluid and controllable jump, as it's always the same height except when you hit your head.

The way the scoreboard and pause menu interacts is also not ideal as it saves the scoreboard two times in each game loop in two different places, but the scoreboard was a bit of a late addition, and it was the quickest way I could get it working.

The game ending function is also not the best as you can start the game again in the background by unpausing the game using ESC. I would find a way to stop the game completely and change the view instead of the QGraphicsScene into the ending screen.

Some parts are also a bit hard coded but I tried to pull all the hard coded things from a setting.py file to make them easier to change if you want to.

Best working feature

The best implemented feature is probably the map creation. New levels are easily created by creating a new map setting file and adding into the list of maps in the setting.py file. New tile types are also easily added if you just write functions for them in the tile class. Obstacles are easy to add, and maps are generated in a way that you can create very different types of maps demonstrated by the first level which is vertical and the second level which is horizontal.

Changes to the original plan

I was very ambitious at the start of the project. Wanted to add animation to the player character, enemies that you could interact with, and more dynamic platforming. Until week 3 I was pretty much on schedule, but the collision system I had was a bit broken still. I really underestimated how long it would take to fix it. After getting stuck repeatedly on the collision detection the motivation for the project really plummeted and paired up with me getting very sick and bedridden for over a week the project came to a stand still for week 5,6,7. Getting sick meant I had to prioritize other courses catching up in other courses which meant slow progress on the project. During the last two weeks I have spent the most time on the project, around 30 to 40 hours. Most of the hard level requirements I have in my program were added on the last week. I also went a bit over the deadline because of the collision detection again.

Realized order and scheduled

I can’t say exact dates or weeks, but this was the implementation order:

Around week 1 and 2: Game loop in which you could move a player around on a basic map, collision detection work began

Week 3: Collision detection

Week 4-5: Collision detection and more work on map

Week 6-7: Pause function, player death, coins, obstacle, moving obstacle

Week 8-9: Hard level things like scoreboard, pause menu, more levels (1 more, but made it so it’s easy to add more, unittests, collision detection fixed to the best of my current ability, clean up of code and win-con added.

Assessment of the final result

I wouldn't say I am very happy with the final result. I think I could have done better if I hadn't lost all motivation for the project. But given the circumstances the final product isn't that terrible. From a gameplay point of view the game isn't that good or fun to play but I have learned a lot from working on this project.

I am glad I added some hard requirement features into the project, even though they were added at the really end. I feel the game could be more interesting if I polished up the movement of the player, for example the jumping that I have already talked about. More interesting obstacles or enemies could be added, I would say it is easy to expand upon the code I have written.

References

https://doc.qt.io/qtforpython-5/contents.html

https://levelup.gitconnected.com/2d-collision-detection-8e50b6b8b5c0

https://learndataanalysis.org/create-a-simple-app-with-qdialog-class-to-print-name-pyqt5-tutorial/

https://www.youtube.com/watch?v=ZVOfc8pZH1A

https://programtalk.com/python-examples/PyQt5.QtWidgets.QMenu/

https://www.youtube.com/watch?v=823ProFM4us

https://zetcode.com/gui/pyqt5/eventssignals/

https://stackoverflow.com/questions/57238032/how-do-i-make-this-pyqt5-character-move-on-arrow-key-input

Attachments

Sorry couldn't get it to work