1. **Personal information**

Title: Tower Defense Game in Python using PyQt5 library

Student's name: Sanzhar Saidnassimov

Student number: 786667

Study program: Digital Systems and Design

Year: 2022

Date: 13.03

2. **General description and difficulty level**

The main objective of this task was to create a tower defense game in Python using the PyQt5 library. The idea of the game is to stop the enemy troops from reaching the exits, by placing defensive structures along the path.

To summarize, the project has fulfilled the medium requirements for the task: the basic game concept has been implemented with a simple graphical interface. To be specific, towers can damage the enemies and eliminate them, while graphics comprises circles and squares. In addition, few unit tests have been created for some parts of the application. Furthermore, conditions for winning and losing the tower defense game have been introduced. Although the program lacks extensibility in terms of custom game configuration, it does have different towers and enemies. An additional feature that has been implemented in the program is the continuous "waves" of enemies. Only two "waves" are present in the code, however, this can be easily extended. Ultimately, it is fair to say that the level of the project is higher than the medium.

3. Instructions for the user

The player sees the game in a PyQT graphical window, which consists of the game map with the pre-drawn enemy path and the tools bar. The tools bar displays the initial balance and lives of the player, towers that the player can purchase, and the tower information. Initially, tower information is not visible, however by pressing any of the tower buttons, the features of the tower will be displayed. Be careful as when any of the towers is clicked, it is automatically selected to be placed on the grid. To de-select the tower, just press at any non-button area on the tools bar.

As can be seen from the tower data, each tower costs a specific amount of in-game currency has a limited range and rate of fire, and the damage they can inflict. The game starts as soon as the player places the first tower anywhere on the map, with the "wave" of enemies starting to follow the path to reach the exit. Do not let the enemies do it by placing the defensive structure along the path. The initial budget is

quite limited, however, each eliminated enemy gains a specific amount of currency. The budget on the tools bar is updated in real-time. The game displays health bars (green lines) of each target, so it is easy to track the health status of the enemy troops. The player has a certain number of "lives", or a number of enemies which can reach the exit. By default, it is 10 (which is quite generous!). If an enemy escapes, the lives of the player are decremented and as soon as it hits 0, the game is over. On the other hand, if all enemies in each wave are eliminated and lives have not yet reached zero, you would be the winner of the game. Current version of the game has 2 waves of enemies, and it is quite easy to win.

To actually start the game, do as instructed in the README file.

4. **External libraries**

To create a graphical interface PyQT Python library will be used. Other external libraries have not been used. The code made use of Python's built-in libraries (*math, sys, time*).

5. **Structure of the program**

All starts from TowerDefense class which is the class for running the game. The game has two sub-linker classes GameBoard and toolsBar. Enemy, Tower and Bullet classes can be part of the GameBoard object. The objects of those classes are stored in the global variables. The basic idea is that TowerDefense class initialized the GameBoard and toolsBar as well as keeps track of the game (monitors all the events that are happening on the map and bar, run the waves, fires the towers, keeps track of time, etc.) The GameBoard class hides all the implementation: drawing all the GUI, objects, updating data. The toolsBar simply displays relevant data to the user.

6. **Algorithms**

To be completely honest, I lack awareness about specific algorithms that have been utilized in my application. However, I can shortly explain where and what my game makes use of to power most of its functions.

The program frequently calculates the center of the coordinate with a simple, yet modified mathematical expressions, to match the scale of the implemented coordinate system. The most important piece of the code is probably the round() method which heavily assists the placement of the tower in the grid.

In terms of enemies, the project incorporates very basic and intuitive path following algorithm, by introducing directions. This simplifies the implementation. For towers Euclidian distance is calculated to check if the enemy is in range of the tower.

Data structures

As predicted in the project plan, the program mostly made use of the lists to store created tower, enemy and bullet objects as well paths and non-occupiable coordinates. However, program also used the *json*-type of files to store information on enemy waves.

Files

As mentioned in Data structures section, the program handles the *json*-type of file. The idea has been inspired by one of the PyQt5 projects from the references. Otherwise, the application does not use any specific types of data, except for the images, which are used as icons to assist with interface.

Testing

Although testing via unit tests was suggested by the course, in the case of the game, the live-debugging and testing by playing the game itself were efficient enough to create final version of the project. In majority debugging was utilized to detect the breaking points in the program, allowing to make necessary tweaks in the code. Multiple bugs have been detected and fixed thanks to debugger. In addition to the mentioned methods, the Python's *print*() command has been very useful to track the values of the variables, finding breaks, keeping track of the functions, etc.

The known shortcomings and flaws in the program

Undoubtedly, the application is very raw and has a multitude of bugs and flaws hidden in it. One of them is the tower placement problem: the towers can be placed on the path itself. Due to lack of time, I was not able to dedicate my attention to the given problem. This is an unpleasant and counter-intuitive shortcoming of the game. The tower itself does not block the movement of the enemies and continues the fire. I think it is a massive shortcoming, as by placing the tower on the road, the player has the most chances to eliminate the enemy. I think I would be able to fix it by moving the path coordinates into the non-occupiable coordinates list.

Another big shortcoming is the enemy's path. I was not quite able to detect the main reason, but the enemy's path is quite unpredictable in the sense that sometimes enemies might not follow the path correctly. Of course, the program uses pretty primitive path-following and with the several variables affecting the movement, some cases of the enemy paths might produce buggy movement.

Otherwise, I did not notice any other major flaws in my program.

3 best and 3 worst areas

Apart from the tower placement and enemy movement implementation, which took a lot of time and thinking (even with the help of the internet and other projects), I am pretty proud of some of the creative decisions that I made in a little span of time. In that way, I am strangely proud of the health status bar implementation. Although it is very simple and is not perfect, I did it as a quick alternative to the QProgressBar which I was not able to employ in my code.

One strong part of the code is the conditional variables. There are so many of them in the code and keeping track was not a simple task, eventually, I managed.

And surely the organized structure of the code. Again, it is not perfect, and can be easily improved, but I did not have major problems when navigating through my code.

Weak parts of the code are repetitive lines. Having more time, I would attempt shortening some code parts into loops. However, some mathematical expressions in the code are also too long and dividing them into smaller parts would help in creating understandable code.

I guess I also can decrease a number of variables that I use, by writing more effective code. For instance, I have some global variables in the constants.py which I think can be avoided.

Changes to the original plan

The majority of the final implementation follows the original plan, although on a less ambitious tone. The change that I noticed is the dragging placement planned initially and the selective placement of the towers in the final code.

Realized order and scheduled

I would not say I followed my plan. I was overwhelmed by other subjects and could not follow the original schedule, making me work long hours in the last week on this project. To be specific, I failed to meet the second programming checkpoint (did not deliver unit tests). I also missed my demo meeting, so had to reschedule it for a second time.

Assessment of the final result

Overall, I believe it is a good program. Undoubtedly, the game has some inconsistencies, however, for the conscious player those would not cause much of a problem. If I had a chance to improve it, I would start by fixing abovementioned bugs and later working on improving other features, including diversifying tower shooting, improving the health bar, diversifying enemy movement and categories. Of course, I could also expand the program by adding the configuring menu.

References

1. https://en.wikipedia.org/wiki/Tower_defenseAttachments
2. https://plus.cs.aalto.fi/y2/2022/project_topics/topics_pelit_105tornip_en/
3. https://github.com/mlisbit/wander-tower-def

Attachments

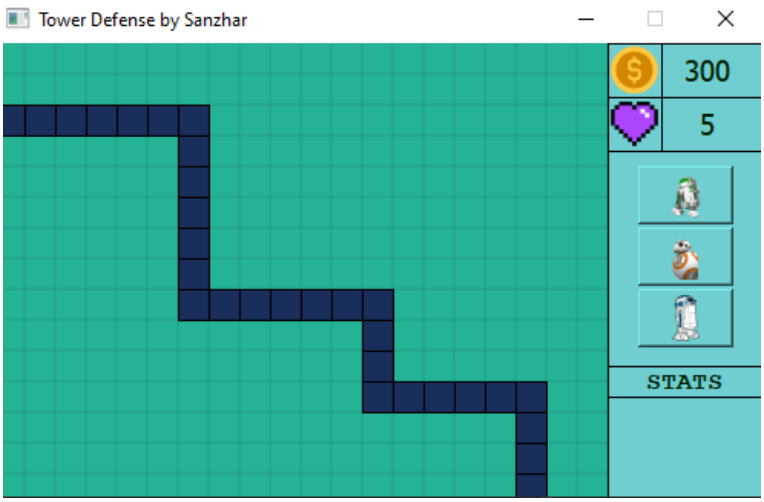


Figure 1: Opening screen of the game: the program waits for the player to place the tower

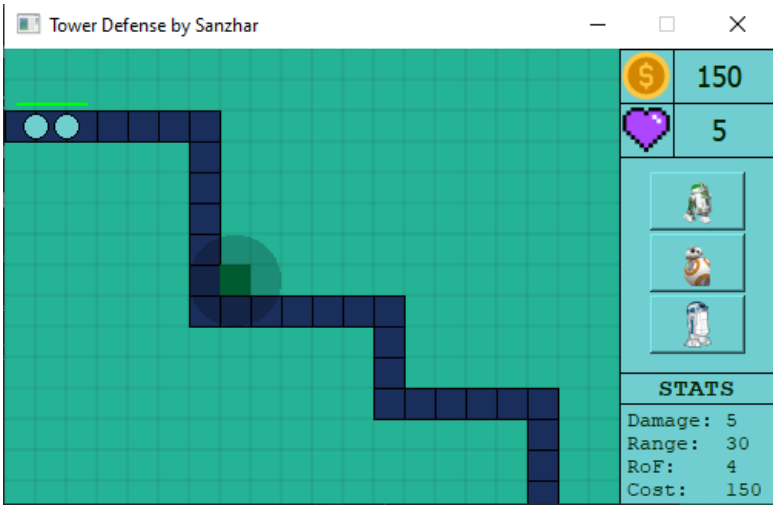


Figure 2: The tower is placed: after a one-second counter, first enemy wave begins

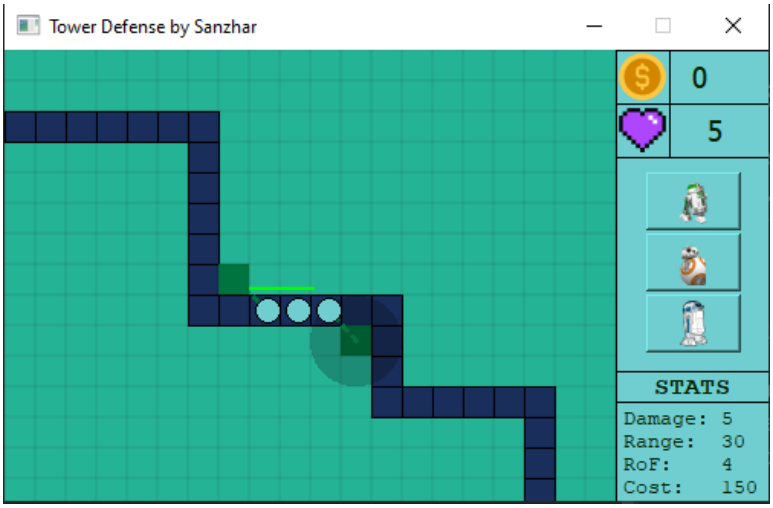


Figure 3: Second tower is placed and both towers are firing at the enemy troops: the health status of the enemies can be seen

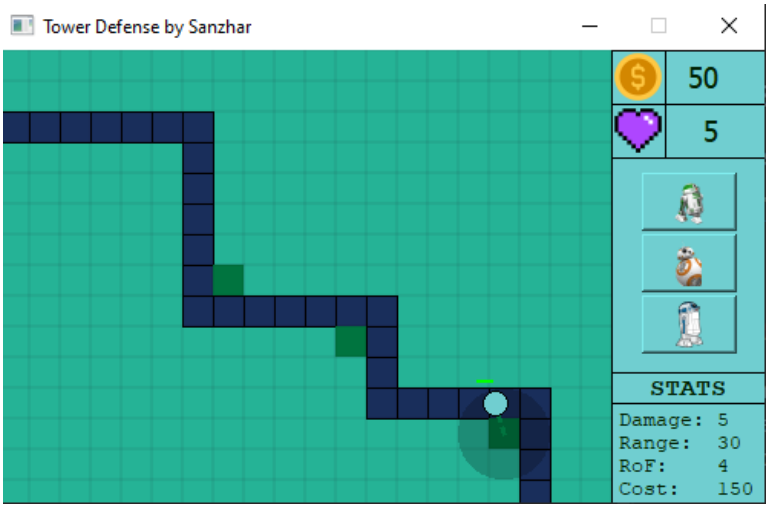
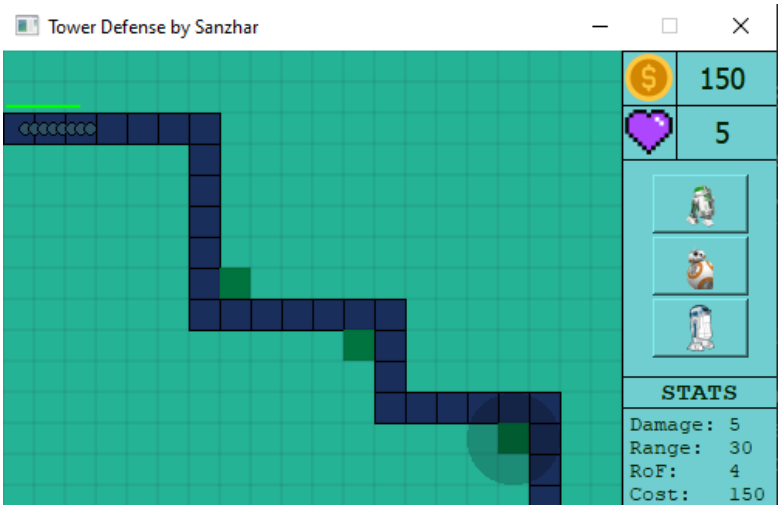


Figure 4: After successfully eliminating 2 targets, enough currency is gained to purchase third tower to finish the final target



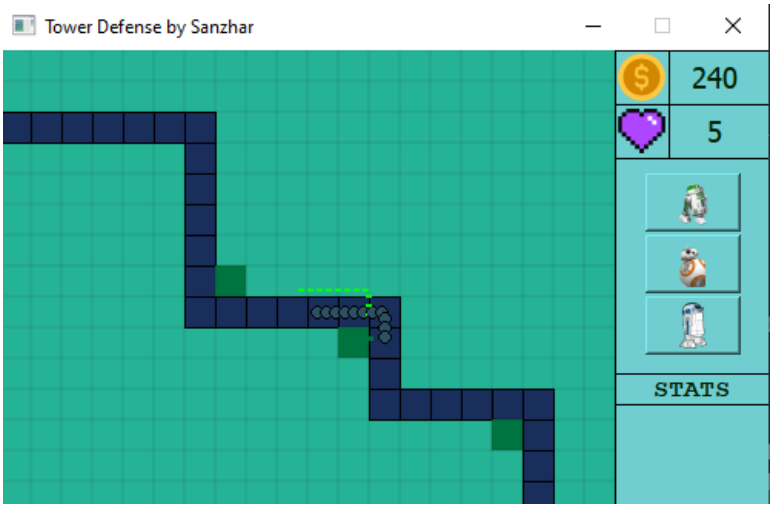


Figure 5 & 6: Eliminating second wave of enemies

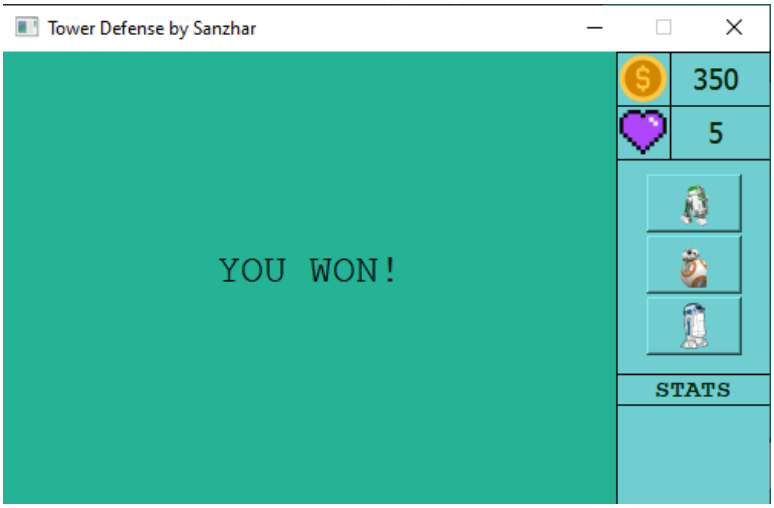
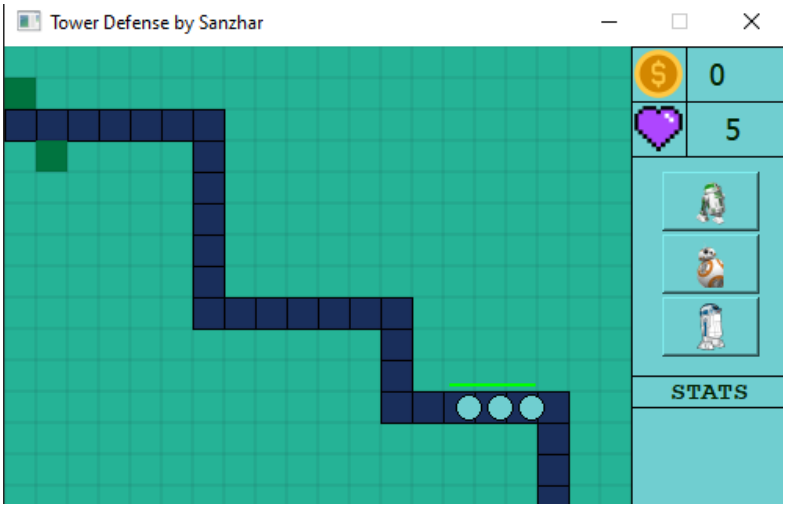


Figure 7: Final result: You won!



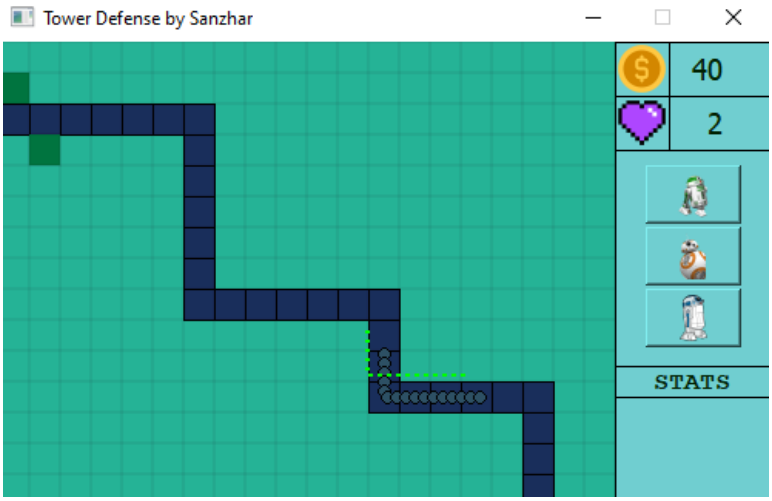


Figure 8 & 9: Poor tower positioning results in enemies escaping and player losing lives

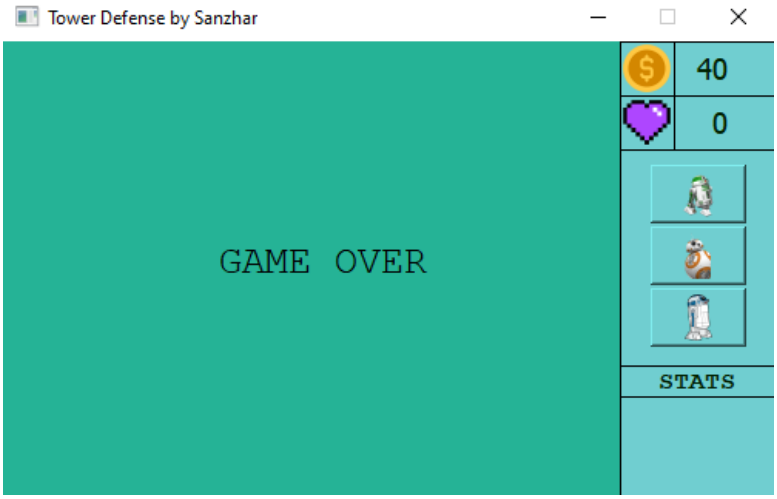


Figure 10: Final result: game over