PROJECT REPORT

CASTLE WAR

RICCARDO RICCIO

Contents:

INTRODUCTION:

Goal and Game dinamics:

The project consists of creating a 2D game using the Pygame library[1].
The game is played in real time by two players sharing the same keyboard and screen, and consists of a battle in which the goal is to destroy the enemy wall using different troops (swordsmen, workers, archers). Each player has a wall that must be protected, a tower that attacks enemy units, a barracks that can train or release troops, and a mine that generates new resources.

DESIGN STRUCTURES AND PROBLEMS:

In order to develop the game from scratch, I divided the process into different phases and then gradually found solutions to implement these phases in the code.

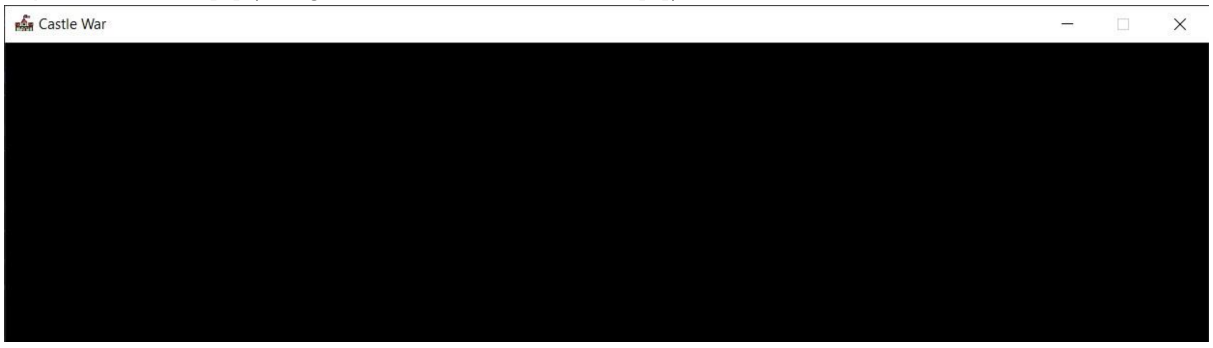
Phases:

1. Create screen, title and icon
2. Load all images
3. Add background and buildings to the screen
4. Create classes for buildings and units
5. Animation
6. Train and rest
7. Dispatch units
8. Visualize writings on screen 9. Visualize occupied barracks:
10. Collision of Units
11. Attack/action of units and towers
12. Visualization of health
13. Visualize the health of units in the same position.
14. Kill units
15. Save game
16. Load game
17. Pause game
18. Win and restart game

SOLUTIONS AND IMPLEMENTATIONS

1. Create Screen, Title and icon

To create the screen, add title and icon, there is a built-in Pygame module that allows easy implementation [2] (image of the icon taken from [3]).



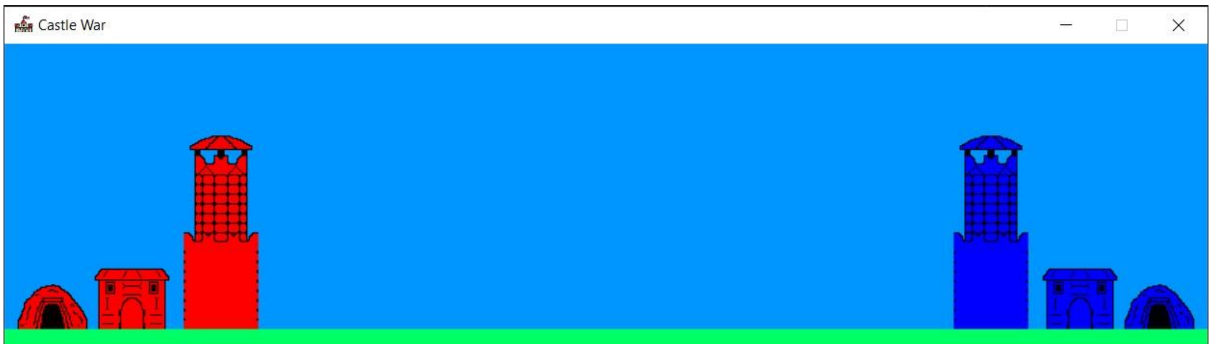
2. Load images

The images used for the project are those given in the exam specification [4].
Since there are a lot of images (more than 100) to load and it would be long and tedious to load them one by one, I used a technique taken from another project [5] (with modified code).

It consists of iterating to all unit folders (worker, sword, archer), then to all unit actions (run, attack, fall) and to all frames (number of images for actions).
Some frames needed for the game were not present in the sprite folder of the test, for example those of the workers running to the mine, and I create them by flipping the normal running frame.

3. Add background and buildings into the screen

The background consists of sky and grass, both created with pygame rectangle[6].
All background images and buildings are placed according to the test specification in [7]



4. Structure of classes for Buildings and Units

To create class the first main design choice was about using different class for different units/buildings or uning a common class. For the buildings i decided to make 3 different classes since they do not share big similarity. For the units instead, since they share similar feature (Health, train cost, speed, etc), I first create a general class (Units) with usefull methods for every type of unit and then create a class for each units using inheritance from the general class.
The idea of using a general class is also useful for possible future updates to the game, which might add new troop types, for example.
Is also useful have different types of classes for units since they have important differences (attack, collision, action, etc.) and need separate methods.

5. Animation:

To animate the units, the main problem was to handle different types of action, multiple frames and to restart from the first frame when the animation reaches the last one. To accomplish this, I used a function from [8] that defines the frame as being composed of a specific action and a specific frame (using lists) and then increments the frame over time and restarts when the last frame is reached.
Another problem on animations was the death animation.
I wanted a unit, when killed, to remain on the ground for some time after the death animation, to do this I created a function that checks if the unit has reached the last frame of the death animation and then starts a timer of one second before removing the unit from the game.

The last problem was the fact that the images change slightly in their y-coordinate after each action change, i solve it by specifing the exact y-coordinate after each action change.

6. Trained and rest

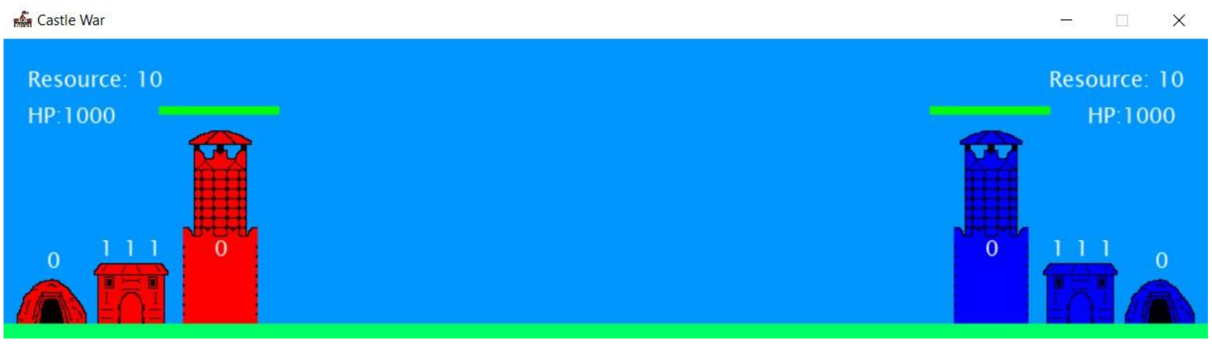
To train a unit, we need to process keyboard commands and available resources.
I create a method ('train_unit') in the barracks class that takes keyboard input (using Pygame methods [9]) from the player, checks if the player has enough resources, if so it creates an instance of the appropriate units and inserts them into two lists, one containing all trained units of a particular type (sword, archer, worker) and a general list 'trained_units'. I decided to use two lists because it will be useful for visualizing barracks units and for collisions.
After training a new unit, the barracks become occupied for a certain time (using a timer) and cannot train other units.

7. Dispatched units

There are two ways of disposition of units: It is possible to dispatch a single unit or all military units, I create two different functions:
The first one ('dispatch_unit') takes keyboard input, checks if units are available in the barracks and if the barracks is not occupied (see next point), if the condition is met, the units are removed from the barracks list and added to another list 'unit_group', which is responsible for the dispatches and living units.
To dispatch all units, the second function ('unleash_all_units') checks for military units after keyboard input and dispatches them one by one using a timer.

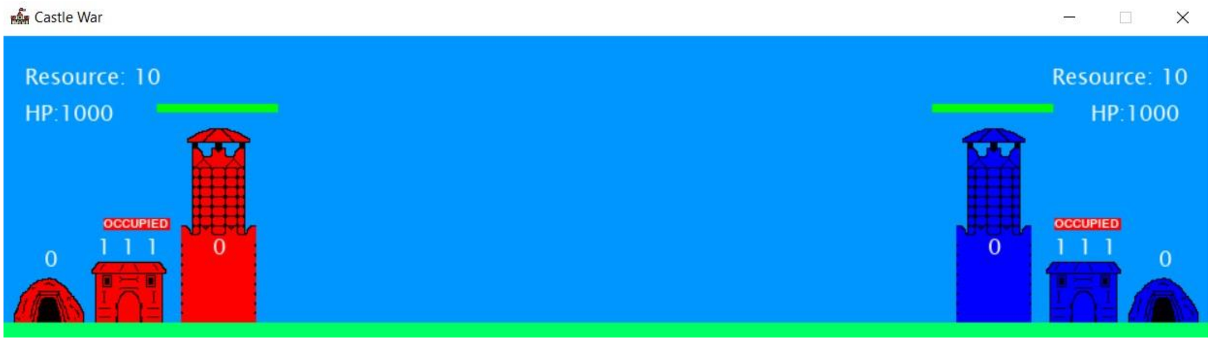
8. Visualize writings on screen

To visualize fonts on the screen, I created a function ('write_on_screen') that takes the message we want to display, the position where we need to display it, and possibly a color or font, and visualizes it on the screen.



9. Visualize occupied barracks

An important addition I made is to visualize a warning indicating to players that the barracks are occupied.
I create a methods (‘barrack_available’) that check if enougth time has passed since last training, during that time it visualize a red rectangle and a writings that states that the barracks cannot be used.



10. Collision

There are several types of possible collisions in the game:
1) Arrows collide with the ground/wall.
2) Arrows collide with units .
3) Collisions between units.
For the first type, I created a function ('ground_collision') that checks if the arrows collide with the ground or the wall (using coordinate), if so, the arrows are deleted.
For the other type, the main problem was to handle the collision so that if different units are in the same position (so that their hit boxes match) and something (enemy/arrows) hits one unit, it does not hit other units, for this reason a simple sprite collision check could not be used.
I solve this problemby assigning a specific target to each attacking unit/arrow so that it can only damage that specific enemy. However, this only partially solves the problem, as various situations can occur, e.g.:
1)   An arrow had a specific target when it was thrown, but that enemy died and the arrow hitanother enemy/wall.
2)   An arrow with a specific target is thrown, but an enemy other than the target is hit beforethe arrow reaches the actual target.
To handle these and other situations, I created a function ('handle_collision') that covers all possibilities: first a queue of enemies is created, then it checks if the units have a target, if not, the target becomes the last element of the queue, if the unit has a target but it dies before the unit can hit, the target is changed and becomes the last element of the queue.

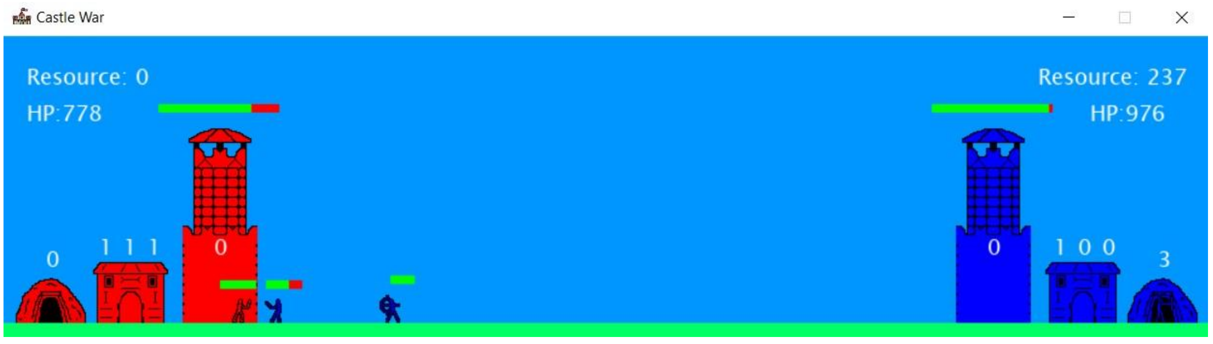
11. Attack/action of units and tower

The action of every units is controlled by specific module of theiry class.
There are different action to manege:
-swordsman/archer : run, attack (and die)
-worker: run, repair wall, generate resource

-units/tower’s arrows: move, hit
To menege run on all the units case it’s simple, we have just to increase/descrease the x coordinate, every turn (that corrispond to one iteration of the main loops, see below) they move of certain space defined by their speed.
To move the arrows thrown by the towers, it is necessary to first determine the angle and then increase/decrease both the x and y coordinates.To solve this problem, I modified a function [5] that shoots a ball into the mouse position, and I adapted it to my case: The new function checks all troops that come into the range of the towers, put them into a queue
 (to get an attack order), finds their coordinates and shoots into that specific point using trigonometry.
Unit attack begins when an enemy enters the attack area (see collision below), units change their action from run to attack, then each time they reach their last frame index of attack (the swordsman uses the sword and deals damage directly and the archer throws an arrow).
Regarding the worker, I created 2 functions ('generate_resource' and 'repair_wall'); the first one makes the worker run into the mine, checks if it has reached it and then changes the action from run to repair, and finally calls a timer that increases the resources by a certain amount per turn. The second function is similar to the first, but refers to the wall.

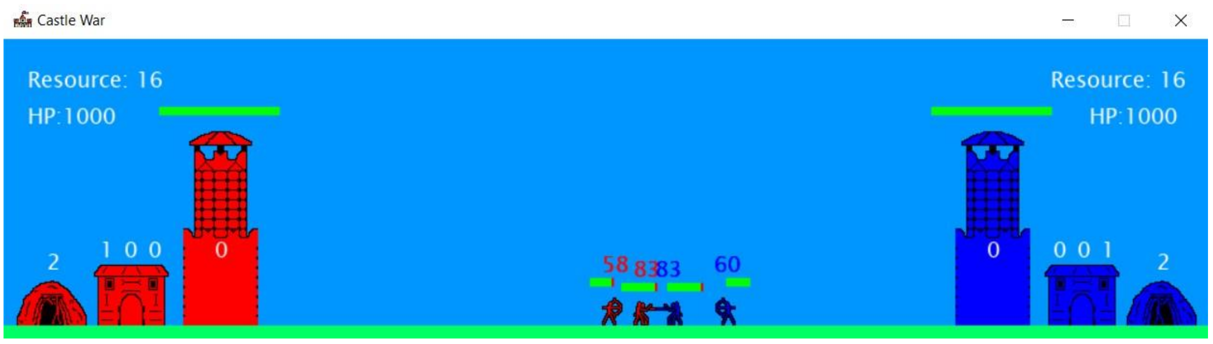
12. Visualization of health

To visualize the health of the units/walls, I decided to use a health bar to help the player understand the state of the units.
To implement the health bar, I created two overlapping rectangles, one red and one green. The red triangle has the length of the unit's maximum health, while the green triangle has the length of the current health, so when an enemy loses health, it becomes smaller and the red rectangle is displayed behind it.



13. Visualize the health of units in the same position

Another problem related to the fact that different units can overlap is that it becomes difficult to know how many units there are and what their health is.
I solved this problem with a function ('units_same_position') that checks if units of the same player are at the same point, sums up the health of all units and displays it on the screen.
Another addition I made is the color of the lettering, because when there are many troops, it becomes difficult to understand if the global health belongs to the units of one player or another.



14. Kill units:

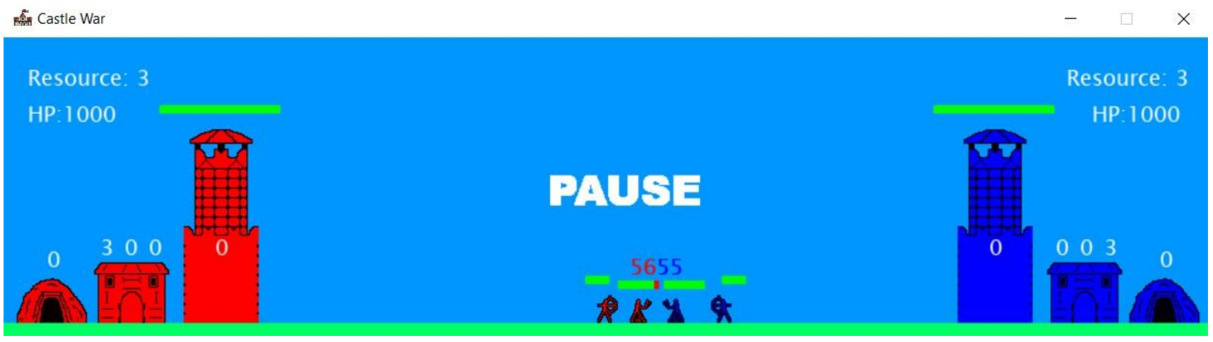
When units reach 0 health, a function ('units_death_animation') is called that first deletes the units from the group of units that can collide so that other players' units do not collide with them, then displays the death animation for one second, and finally deletes the enemy from the screen

15. Initialize game

To initialize the game, I created a function ('initialize_game') that initializes the health of the wall, the resources in the mines, and clears all the lists that manage the game (list of barracks units and units playing), then reinitializes the units in the barracks (one unit for each type). This function is useful to restart a game or load a new one

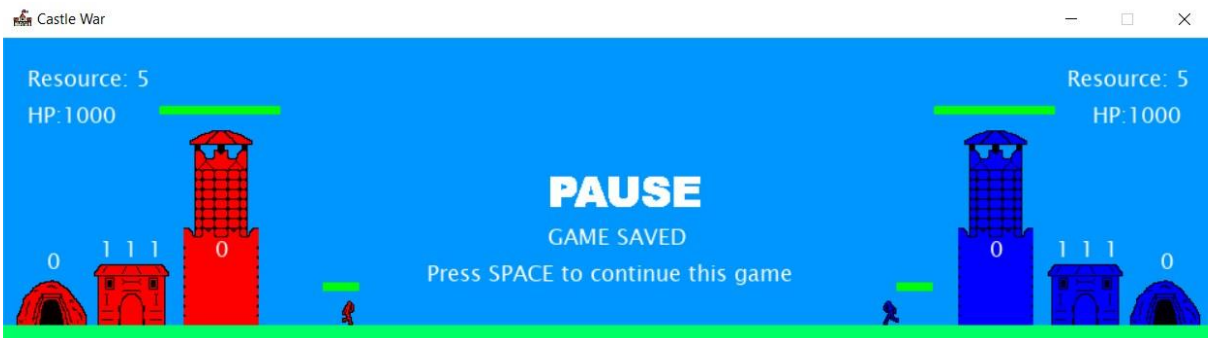
16. Pause game

The pause of the game can be done by both players pressing the ‘SPACE’ key and it’s handle with a variable ‘pause’, that quit the main cycle (see below) in such a way the game stops, then it write on screen that the game has been paused:



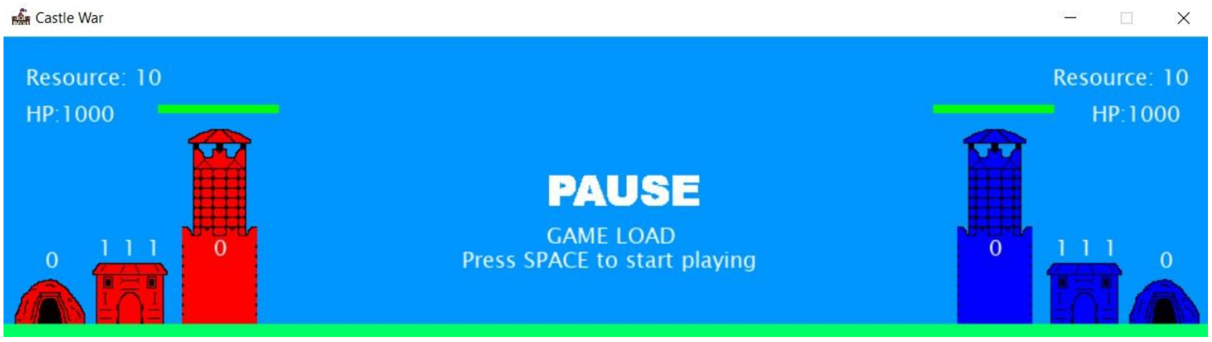
17. Save game

To save the game, I created a function ('save_game') that is called when one of the players presses a certain key ('v'). It creates a Python file and writes all the useful information into it, such as the number and position of troops, the units in the barracks, the health of the walls, and the resources.
Then it pauses the game (see below) and tells the players that the game has been saved.



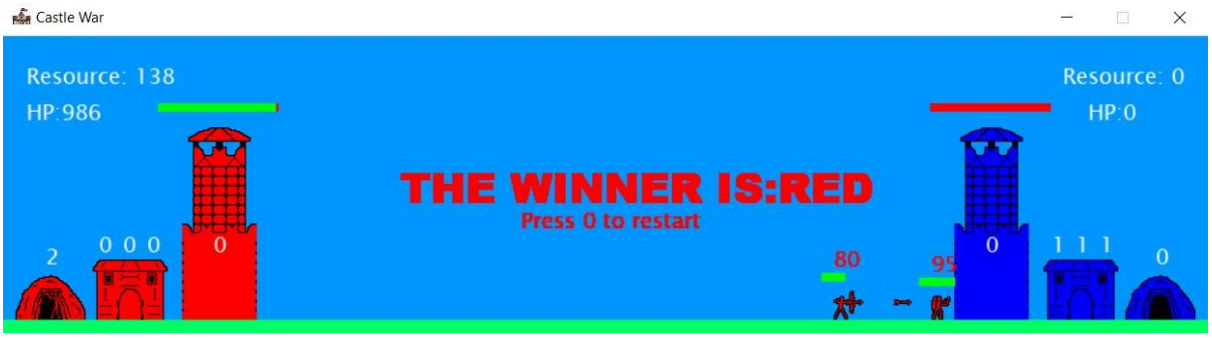
18. Loading the game

Loading the saved game is handled by the('load_game') function, which is triggered when a player presses the 'b' key. It first initializes the gameand then goes to the 'saved_game' file, check the health and resources of the saved game it substituite them to the current ones, then for every units into the barracks (in the saved game) it create new units and add them into current barracks finally for every dispatched units create new units with the same coordinate/characteristics. The load game start in pause (see below) with a writings that warns the the pas game has been loaded.



19. Win

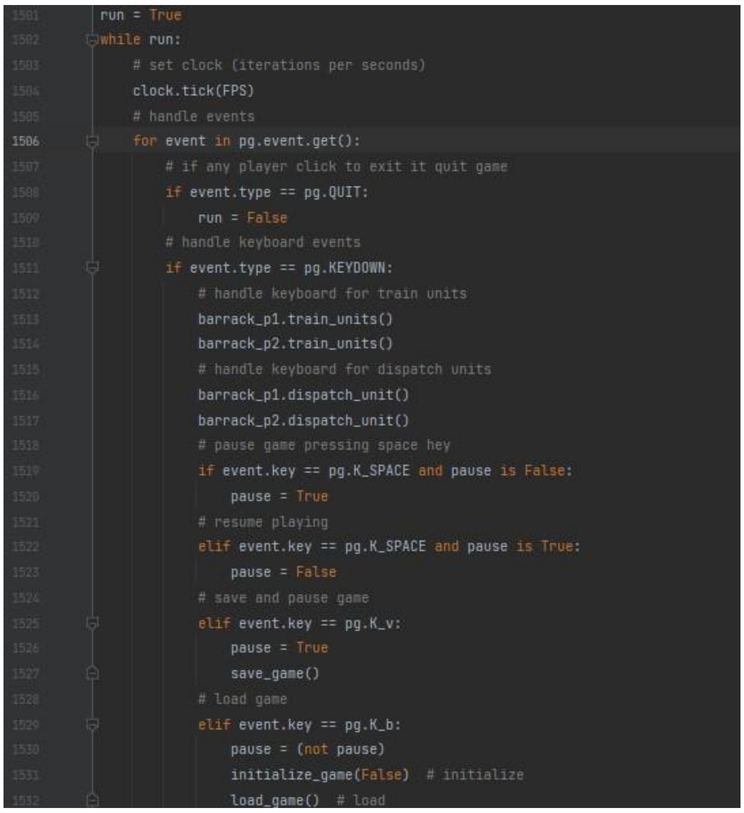
When one of the two walls reached 0 of health the game is stopped by quitting the main loop (see below), this quitting is regulated by a variable that start as True and become False when health reached 0, then on the screen appear a writings that announce the winner with the colour of the winning player and finnaly a celebrating music [10] plays for same seconds. The players have the possibility to start a new game using the key ‘0’.



DESCRITION OF MAIN BLOCKS OF CODE

Main Loop

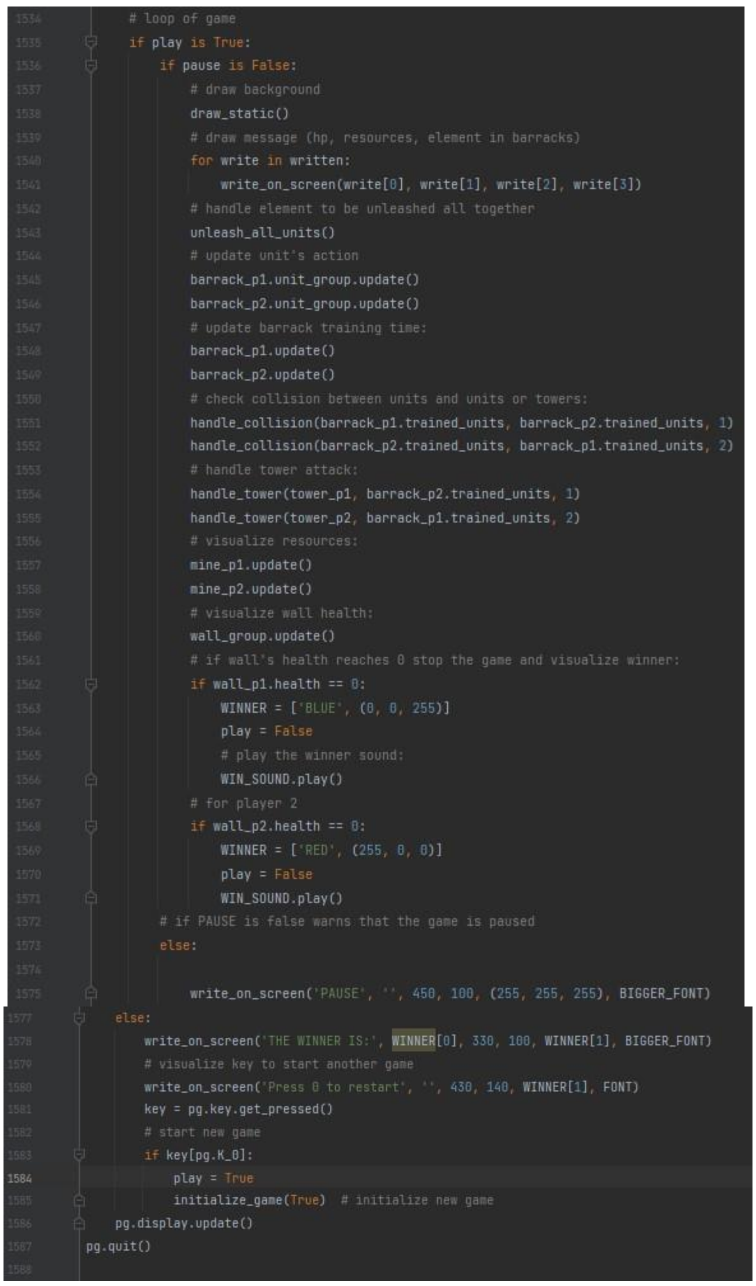
The main part of the code is the main loop that consist of a while loop that continuously update the game, here all the main function (visualize animation, collision, attack, train, distaptched etc) are colled multiple time.The iteration of the loop is regulated by a clock that run 60 FPS for seconds so every iteration, that corrispond to a turn in the game[7], is 1/60 seconds.Now I will describe the code in order to simplify future updating of the game:



```
1501  run = True
1502  while run:
1503      # set clock (iterations per seconds)
1504      clock.tick(FPS)
1505      # handle events
1506      for event in pg.event.get():
1507          # if any player click to exit it quit game
1508          if event.type == pg.QUIT:
1509              run = False
1510          # handle keyboard events
1511          if event.type == pg.KEYDOWN:
1512              # handle keyboard for train units
1513              barrack_p1.train_units()
1514              barrack_p2.train_units()
1515              # handle keyboard for dispatch units
1516              barrack_p1.dispatch_unit()
1517              barrack_p2.dispatch_unit()
1518              # pause game pressing space hey
1519              if event.key == pg.K_SPACE and pause is False:
1520                  pause = True
1521              # resume playing
1522              elif event.key == pg.K_SPACE and pause is True:
1523                  pause = False
1524              # save and pause game
1525              elif event.key == pg.K_v:
1526                  pause = True
1527                  save_game()
1528              # load game
1529              elif event.key == pg.K_b:
1530                  pause = (not pause)
1531                  initialize_game(False)  # initialize
1532                  load_game()  # load
```

-(1502)Start main loop

-(1504)Set clock that handle iterationof the loop -from 1506 to 1585 we handle all pygame events (all things that happen in the game) -1511 to 1532 handle all functions/methods that need keyboard input -1512/1513 manage the training of the units
-1516/1517 manege the dispatching of units
-1519/20 manage the variable pause that when bwcomes false it quit the loop blocking the game -1525/1527 manage the save of current game and stop the game -1529 manage load of saved game, it pause, initialize and call load function



```python
1534        # loop of game
1535        if play is True:
1536            if pause is False:
1537                # draw background
1538                draw_static()
1539                # draw message (hp, resources, element in barracks)
1540                for write in written:
1541                    write_on_screen(write[0], write[1], write[2], write[3])
1542                # handle element to be unleashed all together
1543                unleash_all_units()
1544                # update unit's action
1545                barrack_p1.unit_group.update()
1546                barrack_p2.unit_group.update()
1547                # update barrack training time:
1548                barrack_p1.update()
1549                barrack_p2.update()
1550                # check collision between units and units or towers:
1551                handle_collision(barrack_p1.trained_units, barrack_p2.trained_units, 1)
1552                handle_collision(barrack_p2.trained_units, barrack_p1.trained_units, 2)
1553                # handle tower attack:
1554                handle_tower(tower_p1, barrack_p2.trained_units, 1)
1555                handle_tower(tower_p2, barrack_p1.trained_units, 2)
1556                # visualize resources:
1557                mine_p1.update()
1558                mine_p2.update()
1559                # visualize wall health:
1560                wall_group.update()
1561                # if wall's health reaches 0 stop the game and visualize winner:
1562                if wall_p1.health == 0:
1563                    WINNER = ['BLUE', (0, 0, 255)]
1564                    play = False
1565                    # play the winner sound:
1566                    WIN_SOUND.play()
1567                # for player 2
1568                if wall_p2.health == 0:
1569                    WINNER = ['RED', (255, 0, 0)]
1570                    play = False
1571                    WIN_SOUND.play()
1572            # if PAUSE is false warns that the game is paused
1573            else:
1574
1575                write_on_screen('PAUSE', '', 450, 100, (255, 255, 255), BIGGER_FONT)
1577        else:
1578            write_on_screen('THE WINNER IS:', WINNER[0], 330, 100, WINNER[1], BIGGER_FONT)
1579            # visualize key to start another game
1580            write_on_screen('Press 0 to restart', '', 430, 140, WINNER[1], FONT)
1581            key = pg.key.get_pressed()
1582            # start new game
1583            if key[pg.K_0]:
1584                play = True
1585                initialize_game(True)  # initialize new game
1586        pg.display.update()
1587    pg.quit()
1588
```

-1535 the play variable is True and when become false it will quit the loop blocking the game -1536 pause variable handle pause -1538 visualize background and buildings -1540/41 manage the writings of the units in the barracks -1543 manage the function that unleah alll units together -1545/46 use with built in pygame sprite groups[9] to update all playing units actions -1551/52 menage collision of units 1554/55 menage the action of the towers
-1557/58 visualize resources in the mines
-1560 visualize wall's health -1562/71 check if health of one walls reached 0, create a variable that rappresent the winner, stop the game leaving the loop and start the celebrating sound.the game leaving the loop and start the celebrating sound.
-1577/80 when th game is stopped, the winner is displayed -1581/84 handle the restar of the game
-1586 update the screen (everytrun)
-1587 quit the screen

TESTING AND TOOLS

The writing and the testing of the game was done with using pycharm; it disposes of an internal debbuger and it can recognize different type of error: syntactic, logical and grammar error.

RESULTS AND POSSIBLE IMPROVEMENTS

The goal of the project was successfully achieved and all the main problems (such as:

keyboard operation, attack, collision, visualization, etc.) were adequately solved, the game does not have any bugs or problems that affect the game experience.
Various useful additions have been made, such as health bars, visualization of health of multiple units at one point, visual indication of occupied barracks, management of all collision strips and others
The game can be improved in different ways, for example:
- Addition of other types of units (i create a common units class on pourpuse) or add diffferent type of action.
-Adding a menu at the beginning can be useful in different ways, it can have a button to show a tutorial explaining the rules/keyboard, or it can add a button to see all previous saved games and it makes the game look better.
-The addition of other music input (other than the celebrating win) such as soundtrack or the sound of attacking/dying units can enhanced the game experience.

REFERENCES

[1] pygame website: https://www.pygame.org/news
[2] screen, icon documentation :
https://www.pygame.org/docs/ref/display.html
[3] Icon image = https://www.flaticon.com/search?word=castle
[4]
[5] load image = http://www.codingwithruss.com/
[6] pygame rectangle = https://www.pygame.org/docs/ref/rect.html
[7]
[8] animation function = http://www.codingwithruss.com/pygame/shooter/player.html
[9] handle key = http://www.pygame.org/docs/ref/key.html
[10] pygame audio = https://www.pygame.org/docs/ref/mixer.html
[11] pygame grups of sprite = https://www.pygame.org/docs/ref/sprite.html