TD Game Documentation

Demo Video: https://youtu.be/GLkcOq6tYJk

Content

Description

Since this project consists of different classes, it is better to separate different components so that it is easier to manage the whole project. In view of this, our program is separated into 5 main parts: MainWindow, GameMap, Player, Enemy, Tower.

MainWindow

MainWindow is used to handle most of the game logic, as well as the implementation of the GUI. Due to the nature of Tower Defense, we also separated our main window into different parts. They will be used to display the gameplay, the tower statistics as well as tower shopping.

GameMap

It is important for us to separate the game map from the MainWindow logic flow because it is also used to handle the rendering of map terrain, enemies, towers, health bars, etc. In order to increase the performance of our game, we opt to separate the GameMap from the rendering of the Enemy GUI.

Player

Player class is used to store and manage the information of the player (defender). Such as money, number of towers, towers owned. This class is also used to access all the towers on the map, as well as how much health is left from the player.

Enemy

Enemy class is used to store the information of the enemy (attacker). This includes their health, speed, gold (how much money a player gets when being killed). It also stores the location which will be used to render the enemy unit on the game map.

Tower

Tower class is an Abstract Base Class (ABC) which is used to store the information of a tower. This class provides a blueprint, such as the cost, upgrade_cost, damage, attack range, etc, for its derived class. In our project, we have 2 derived classes from Tower, Arrow and Cannon. In the derived classes, we will store different statistics (such as damage, attack range, attack speed) depending on the nature of the towers. By implementing the Tower class as an ABC, we can easily manage the derived classes' (towers') statistics when creating new towers.

Tower Types

In our game, we have 3 different types of towers: Arrow, Cannon and Wizard. Arrow tower attacks quickly, but has a short attack range and does less damage. Cannon attacks slowly, but has a longer attack range and does more damage. Wizard is a special tower type which can attack multiple enemies at the same time. The number of enemies to attack will depend on the level of the tower. All towers have 3 stages depending on the level of the tower. For level 1-5, they will be in stage 1. For level 6-9, they will be in stage 2. For level 10 (max level), they will be in stage 3. Different stages will have different appearances.

==========

Arrow: Stage 1:  Stage 2:  Stage 3: 

==========

Cannon: Stage 1:  Stage 2:  Stage 3: 

==========

Wizard: Stage 1:  Stage 2:  Stage 3: 

==========

Enemy Types

In our game, we have 3 different types of enemies: Armour, Scout and Warmachine. Armour is an enemy type with balanced health, moving speed. Scout is an enemy type which has a very fast moving speed, but has very little health. Warmachine is a heavy armor machine which has a lot of health, but moves very slowly. Their statistics will increase as the wave increases. The enemies also have 3 different stages depending on the number of waves.

==========

Armour: Stage 1:  Stage 2:  Stage 3: 

==========

Scout: Stage 1:  Stage 2:  Stage 3: 

==========

Warmachine: Stage 1:  Stage 2:  Stage 3: 

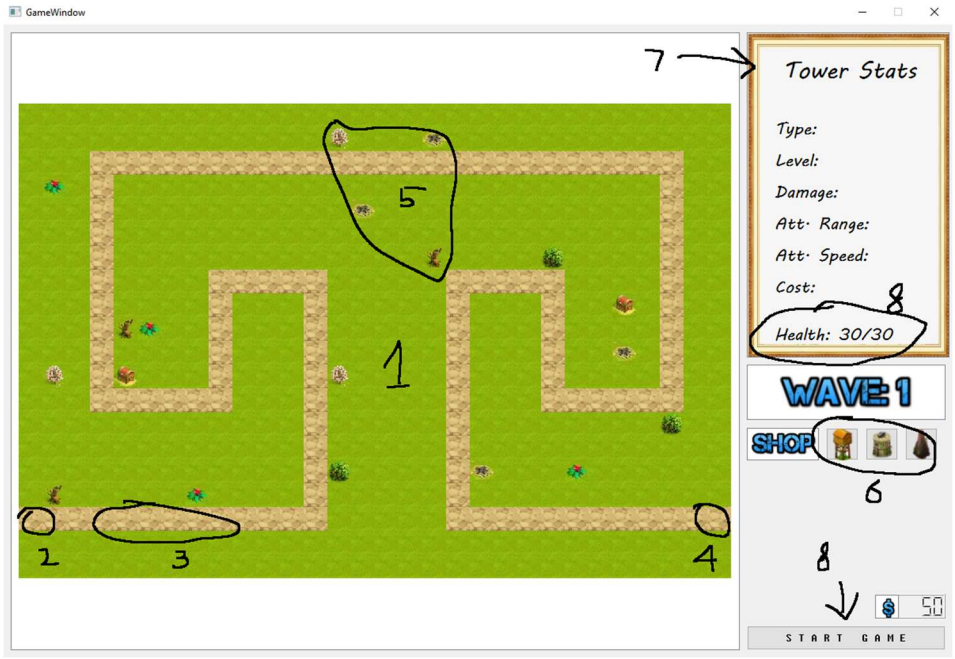
==========

At the beginning of the game (wave 1), all of the enemies will be on stage 1. Starting from wave 6, all of the enemies will enter into stage 2. They will have a different look, and have more health, higher moving speed, and carry more gold. And in the final wave (wave 10), they will enter stage 3. However, even though the enemies are in the same stage, their

statistics will depend on their level (which depends on the current wave). Which means the switching of stage is only a visual difference.

Game Control

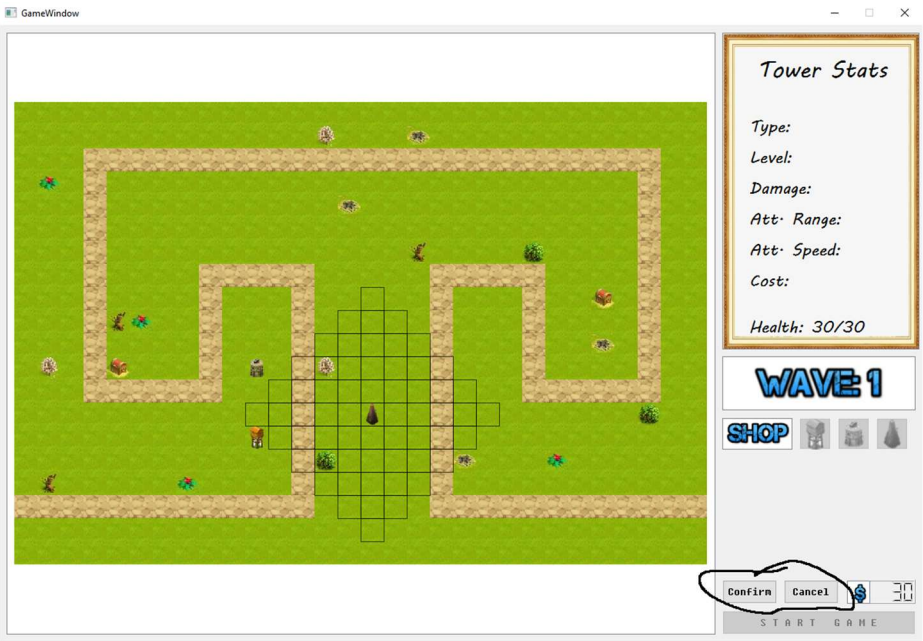
In this Tower Defense game, the player will be greeted with an interface like this:



In the picture shown above, area marked (1) is where you can place your towers. The maximum number of towers you can build is currently 80, which can be changed easily inside the program. (2) is where the enemies will spawn. After spawning the enemies, they will follow the path indicated by (3), until they reach 4 (end path). If an enemy reaches the end, the player's health (shown in (8)) will decrease by a certain number depending on the type of the enemy.

At the start of the game, you will have $50 to build towers. You have 3 tower types to build: Arrow, Cannon and Wizard. Arrow has a faster attack speed but does less damage, while Cannon has a slower attack speed but does more damage. Wizard is a special tower which can attack multiple enemies at the same time (the number of enemies that the tower can attack depends on the level of the tower). Before the game starts (or after each wave ends), you are allowed to allocate your tower and build different towers as you like. You can

either choose the 3 towers in (6) to build, or upgrade the existing towers. (5) are the decorations on the map. It will be automatically generated each time you open the map.



After clicking the tower on the map, users will be able to see the attack range of the tower, as well as prompted to confirm/cancel the build.

When clicking an existing tower on the map, the user will be able to see the tower attack range indicated on the map. You can also see the tower statistics on the top right corner. On the bottom right corner, users can choose to upgrade/destroy the tower as well.

After clicking the “START GAME” button on the bottom right, the wave will begin. The enemies will follow the path to go to the end. When the enemies are in the tower’s attack range, they will be attacked. If a tower kills an enemy unit, the player’s gold/money (shown on the bottom right) will increase by a number depending on the type and level of the enemy unit. If the enemy survives and manages to get to the end alive, the player’s Health (seen on the middle right) will be decreased by a number depending on the types of the enemies.

In our game, there will be 10 waves, when one wave ends, the game will move to the next wave and the user will be allowed to build towers to set up for the next wave. As the wave goes up, the enemy will also possess more health, more speed and carries more gold.

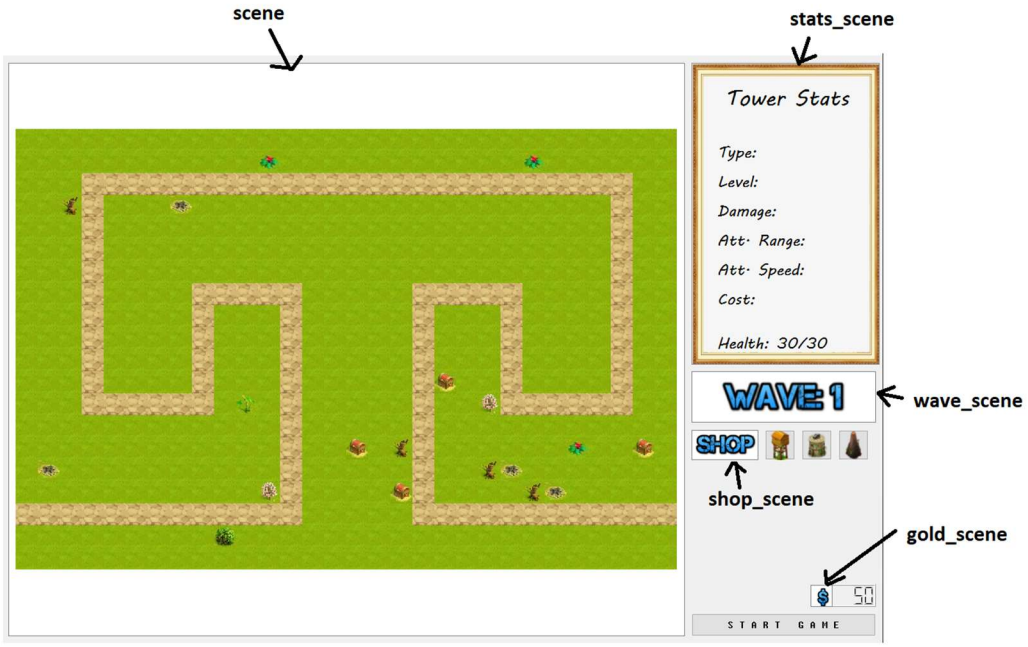
Implementation

MainWindow

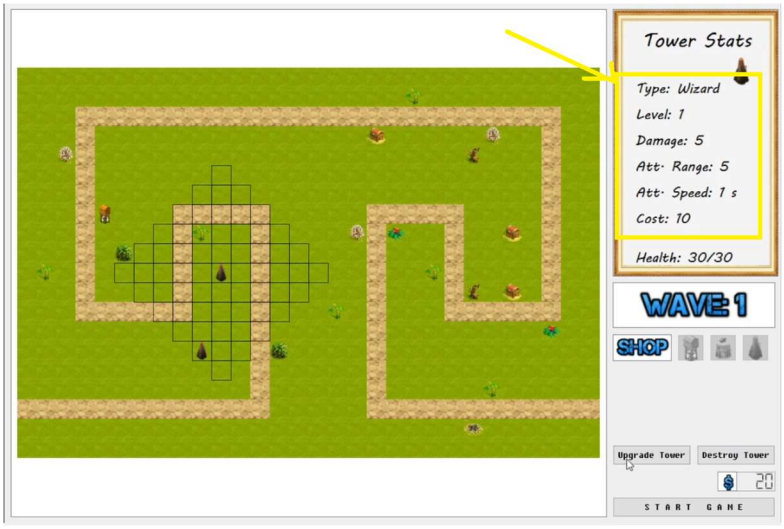
MainWindow class is the class that handles the UI and all game logics. The main member variables and member functions are stated as below:

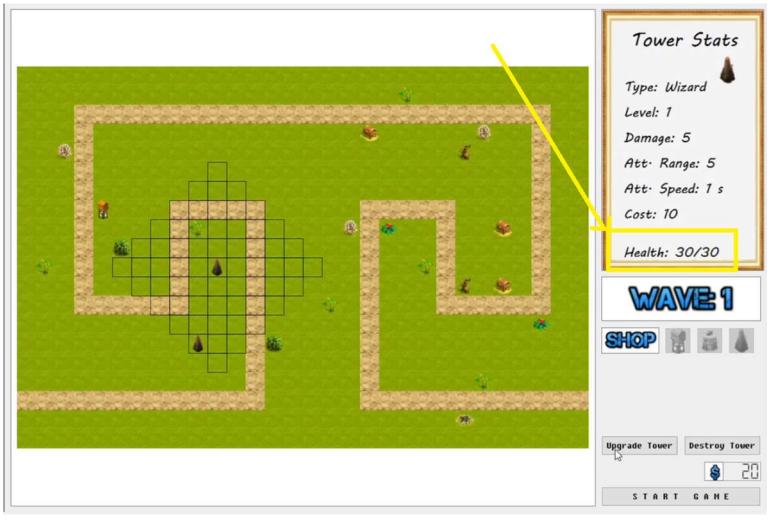
Variables:

- `GameMap game_map`
    - Used to create a new game map
- `Player* player`
    - Used to create new player for the game
- `int current_wave, max_wave`
    - The current wave of enemy and the final wave.
- `QList<Enemy*> enemies_on_map`
    - List that stores enemies which are on the map now.
- `QList<Enemy*> enemies_clock`
    - List that stores enemies which are ready to spawn.
- `QTimer* spawn_clock`
    - Timer to spawn enemy in a certain time interval determined by the current wave.
- `QTimer* game_clock`
    - Timer to handle different checking of the following game state:
        - Enemies that are inside towers' attack range.
        - Enemies' life state.
        - Player's health
- `QGraphicsScene scene, wave_scene, gold_scene, shop_scene, stats_scene`

- These are the scenes referring to different parts of the game UI
- 

- `QGraphicsPixmapItem* wavePixmap, goldPixmap, shopPixmap`
  - These are pointers pointing towards the pictures that are displayed in the corresponding scenes. This pointer is used to change the picture during the game, let's say, the WAVE image.
- `QGraphicsPixmapItem* stats_displayed_tower`
  - This is a pointer which points toward the Pixmap that stores the picture's directory. This pointer will be used to adjust the picture as the user select different towers on the map.
  - 

- `QList<QGraphicsTextItem*> stats_text`
    - This is a QList storing pointers which points toward the TextItem for the displayed statistics of the towers. This pointer will be used to adjust the text during the game.
    - 

- `QGraphicsTextItem* health_display`
    - This is used to store the TextItem used to display the player's health
    - 

Functions:

- `void on_startButton_released();`
    - Start the game by initializing the spawn_clock and game_clock.

- `void map_clicked(int row, int col);`
    - Handling the following map clicking events:
        - Building tower.
        - Checking tower status.
- `void count_down();`
    - Function that is called by the game_clock.
- `void spawn_enemy()`
    - Function that is called by the spawn_clock.
- `void start_rendering_enemies_gui()`
    - A function that starts to render enemy GUI by first creating the PixmapImage and then rendering the enemy movement.
- `void game_end(bool win=true)`
    - Function that handles the end game and calls a messageBox.
- `void end_round()`
    - Function that proceeds the game to the next round.
- `Enemy* generate_random_enemies(int idx)`
    - Function called in the spawn_clock that generates a random enemy.

GameMap

GameMap class is the class that stores the geometry of the map and GUI of the enemies and towers. The main member variables and member functions are stated as below:

Structure:

- `enum class TerrainState{SPACE, UP, DOWN, LEFT, RIGHT, BLOCK}`
    - `SPACE` refers to free spaces for building towers.

    - `UP, DOWN, LEFT, RIGHT` refer to enemies' paths which indicate the enemies' direction.
    - `BLOCK` refers to places that are not available for building towers.
- `enum class EnemiesTowersState{SPACE, TOWER, ENEMY}`
    - `SPACE` refers to places without enemies or towers.
    - `TOWER` refers to places with towers.
    - `ENEMY` refers to places with enemies.

Variables:

- `const int num_rows, num_cols`
    - The total number of rows and columns in the game map.
- `const int spawn_row, spawn_col`
    - The spawning location of the enemies.
- `const int goal_row, goal_col`
    - The destination of the enemies. If the enemy reaches the goal, player health will be decreased.
- `TerrainState* terrain_map`
    - An array storing `TerrainState` of each game grid.
- `Enemies_towers_state* tower_and_enemies_map`
    - An array storing `Enemies_towers_state` of each game grid.
- `QList<QGraphicsPixmapItem*> map_images`
    - A `QList` storing map images of each game grid.
- `QList<QGraphicsRectItem*> map_indicators`
    - A `QList` storing map indicators of each game grid to show tower attack range.
- `QList<QGraphicsPixmapItem*> tower_images`

    - A `QList` storing tower's images of each tower.
- `QList<QGraphicsPixmapItem*> enemies_images`
    - A `QList` storing enemies' images of each enemy.
- `QList<QGraphicsPixmapItem*> health_images`
    - A `QList` storing enemies' health bar images of each enemy.

Functions

- `void render_map_gui(QGraphicsScene &scene)`
    - Render the image of each game grid.
- `void render_tower_gui(QGraphicsScene &scene)`
    - Render the image of all towers.
- `void render_enemy_gui(QGraphicsScene &scene, Enemy *enemies_on_map)`
    - Render the image of an enemy.
- `void render_enemy_movement(Enemy *enemies_on_map, int idx)`
    - Render the movement of an enemy by updating the offset.
- `void indicate_area(int row, int col, int rad)`
    - Render rectangular grey boxes for indicating the attack range of the tower.

Tower

Tower class is an Abstract Base Class. The main function of this class is to provide a blueprint for its derived classes. In our game, we have 3 derived classes inherited from Tower class: Arrow class, Cannon class, Wizard class. They have different stats, and some have special skills. The main member variables and member functions are stated as below:

Structure:

- `Enum class TowerType {ARROW, CANNON, WIZARD}`

- Refer to the type of the tower.

Variables:

- `TowerType tower_type`
  - The type of the tower (Arrow, Cannon, Wizard).
- `Int tower_cost, tower_upgrade_cost`
  - The cost for building and upgrading the tower.
- `Int tower_level, tower_damage, tower_range, int tower_attack_speed`
  - Statistics of the tower.
- `Int position_row, position_col`
  - Coordinations of the tower on the game grid.
- `Virtual QPixmap get_tower_pixmap() const`
  - This is a virtual function inside the Tower class. The usage of this function is for the GameMap to retrieve the image to be shown on the screen. Since the tower will have different appearances when they reach different levels, so the image will also need to be changed. This function will be overrode by the derived classes. For example, Arrow tower will return a Arrow tower pixmap. And as tower level increases, it will return a different Pixmap to the GameMap to be displayed.
- `Virtual bool tower_level_up()`
  - This is a virtual function inside the Tower class. Since different types of tower have different statistics, the level up function needs to be handled separately in the derived classes. This function will also return a bool value indicating whether the level up (upgrade) is successful or not. This function will be overrode by the derived classes.

- `Virtual QString tower_name_string() const`
    - This is a virtual function inside the Tower class. It will return a QString storing the name of the tower. This is used to display the name of the tower in the top right corner of the game. In MainWindow, when we update the Tower Stats GUI, this function will be called. For example, Arrow tower will return a "Arrow" string. Since different types of tower need to return a different QString, this function needs to be handled separately in the derived classes. Because of this, this function will be overrode by the derived classes.
- `QTimer* attack_timer`
    - A timer that calls the attack function in a time interval determined by the attack speed of the tower.
- `QList<Enemy*> current_focus_enemy`
    - A list that stores enemies that are inside the tower attack range.

Derived Classes:

- `Const int _tower_attack_damage[10], _tower_attack_range[10], _tower_attack_interval[10]`
    - Arrays that store the tower's attack, attack range and attack interval with corresponding level.

Tower's derived class: Arrow, Cannon and Wizard

Arrow, Cannon and Wizard are 3 classes which inherits Tower class. By deriving these classes, we can easily manage the statistics, as well as adding new abilities to new towers. For example, we change the tower's attack speed, attack range, etc in the class. We also added a new ability for the Wizard tower (attacking multiple enemies) by implementing it on the Wizard class.

Variables:

- `Const int _tower_attack_damage`
    - This is an array that stores the attack damages of the tower depending on their level
- `Const int _tower_attack_range`
    - This is an array that stores the attack ranges of the tower depending on their level
- `Const int _tower_attack_interval`
    - This is an array that stores the attack speed (or interval) of the tower depending on their level

Functions:

- `Virtual bool tower_level_up() override;`
    - This function will override the virtual function in the Tower class. Please refer to the above section (in the Tower class) for explanation
- `Virtual QPixmap get_tower_pixmap() const override;`
    - This function will override the virtual function in the Tower class. Please refer to the above section (in the Tower class) for explanation
- `Virtual QString tower_name_string() const override;`
    - This function will override the virtual function in the Tower class. Please refer to the above section (in the Tower class) for explanation

Enemy

Enemy class is the base class of all enemies. The main member variables and member functions are stated as below:

Structure:

- `Enum class EnemyType {SCOUT, ARMOUR, WARMACHINE}`
    - Refer to the type of the enemy.

Variables:

- `Int level, health, max_health, speed, gold`
    - Specs of the enemy.
- `Int position_row, position_col`
    - Coordinations of the enemy on the game grid.
- `QGraphicsPixmapItem* image{nullptr};`
    - The PixmapItem of the enemy image.
- `QGraphicsColorizeEffect* effect;`
    - A ColorizeEffect that acts as a hit indicator.

Functions:

- `Void receive_damage(unsigned int damage)`
    - Called when enemy units are attacked, and a check is run constantly to see if units are dead.

Derived Classes: (Armour, Scout, Warmachine)

The three classes are similar in design and are classified in one catalogue for convenience.

- `Const int level_health[10], level_attack[10], level_gold[10]`
    - An array of health, attack and gold for 10 levels of units, as enemy units get stronger each wave.
- `virtual QPixmap get_enemy_pixmap() const override;`
    - This overrides the virtual function in the enemy class, and displays different pictures on the map for different enemies in different waves.

Player

Player class is a class that stores all the towers on the map.

Variables:

- `Int money`
    - Number of money the player currently have
- `Int num_tower`
    - Number of towers the player currently have
- `Int health`
    - The player's current health
- `Int max_health`
    - The maximum number of health the player can have
- `Int max_tower`
    - The maximum number of towers the player can build
- `Tower** towers`
    - A pointer pointing to a list of towers that the player owns.
- `Int max_tower`
    - The maximum number of towers the player can build

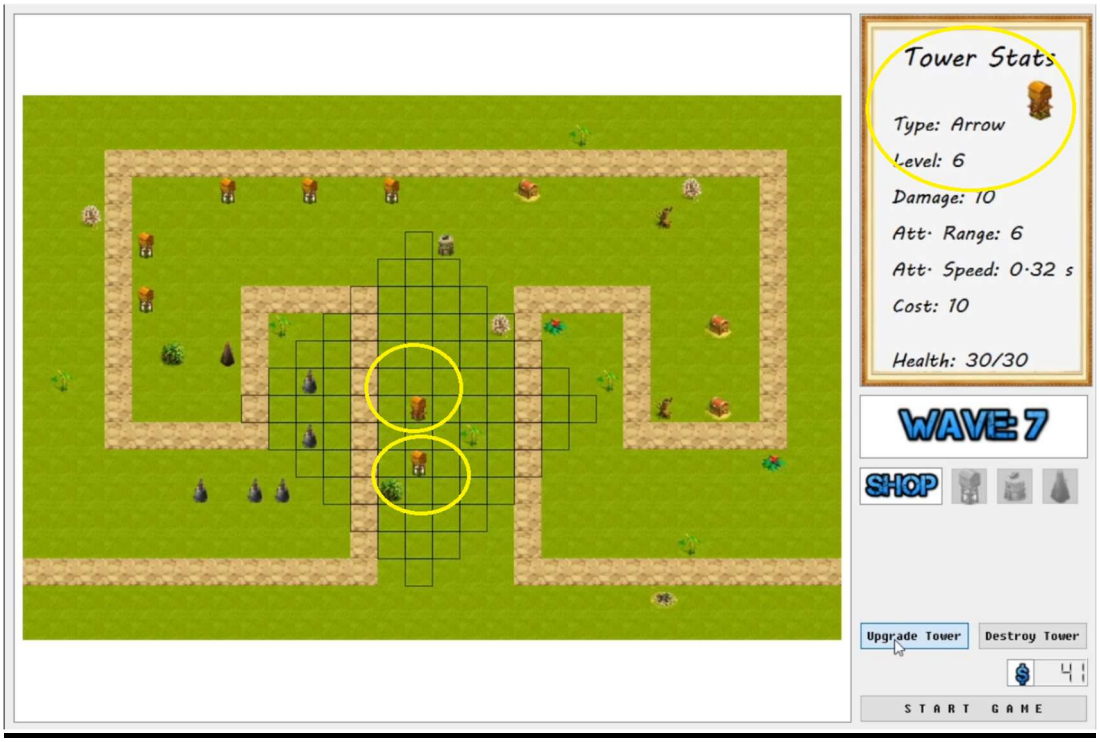
Functions:

- `Tower** get_towers() const`
    - Return the list of towers that the player owns
- `Void destroy_tower(Tower* tower)`
    - To remove the tower stored inside player's towers list
- `Void place_tower(int row, int col, Tower::TowerType tower_type)`
    - To add a tower to the player's towers list.
- `Void reset_focus()`
    - To access the tower list in the player object, and clear the list that stores the current enemies that are in the tower's attack range.

Extra Albums

Wave 6, enemies appearances changed:

Tower at level 6 (as well as level 10), appearance will change



Wave 10 (Last wave), enemies' appearances changed once again. (and a lot more enemies will be spawned):

Game End screen: