PHASE 1

Brief Introduction

KU Tower Defense is a strategy game which belongs to (non surprisingly) tower defense genre[1]; the goal is to defend the map by preventing the enemy attackers from reaching the exit at the end of the path. This goal is expected to be achieved by building defensive towers at pre-determined locations along the path of attack; in turn towers fire different types of projectiles which follow and hit enemies within a certain radius. Each enemy reaching the exit of the map will cause the player to lose one hit point (represented with hearts) and if a player lose all his/her hit points they lose the game. Enemies appear in waves at the starting point of the path on map and follow the path all the way to the exit from the map, unless they are stopped earlier.

You'll be provided with a graphical asset pack for the game; consider the illustrations on the following pages and the provided graphics asset package just as a guideline; do not let them limit your creativity and artistic liberty, you don't have to follow them

[1] some of the most popular games belonging to this genre are Kingdom Rush series, Bloons TD series, Plants vs Zombies series and GemCraft series

pixel by pixel. Feel free to use your own graphics/audio and enhance the visual design of the game as long as you stay close the general theme of the game idea.

1. Gameplay:

1.1 Interaction

Player interact with the game through mouse clicks, player can click on tower slots to build new towers, on towers to upgrade or sell it, on dropped items on the path to collect them, on icons to select abilities, and also on any point through the path to use a selected ability at that location (some of these game play mechanics will appear on the 2nd phase of the game).

2. Game objects:

2.1 Enemies

There are two types of enemy in the game; the goblins and the knights (you expected we would be the knights don't you?). The goblins are faster than knights, they are weaker against arrows due to not having an armor but stronger against spell tower attacks due to their arcane nature.

The knights move slower than goblins, they are stronger against arrows due to their armor but weaker against spell attacks because of their human nature. Enemies (neither goblin nor knight) do not attack the towers directly, their objective is to complete the course and arrive to the end of the path, which in turn causes the player to lose one hit point.

As mentioned previously, enemies appear in waves at the starting point of map and run through the path to the exit. Each wave consists of groups, groups of a wave appear one after another separated by some short delay and without waiting for the previous group to be eliminated first. Waves too appear one after another but waves are separated by a longer delay. The game ends when either the number of hit points of the player hits zero, or when all waves of enemies are defeated.

When an enemy is defeated the player automatically obtains some gold which he/she can use to construct new towers on predefined spots of map or upgrade the ones which are already constructed.

2.2 Towers

There are three types of towers in the game, the archer tower, the artillery tower, and the mage tower.

The **archer tower** attacks with arrows, each arrow targets a single enemy. The rate of fire of archer towers is higher than both the artillery and mage tower. There is nothing more terrifying than the whistling sound of an arrow passing right by your ear.

The **mage tower** attacks with magic spells, each spell targets a single enemy. The rate of fire of mage towers is lower than archer tower but still higher than artillery tower.

The **artillery tower** attacks by throwing an artillery shell to the picked enemy, when the artillery shell hits the ground it causes an AOE (area of effect) damage which affects all enemies within a certain radius of impact.

Towers can only be constructed to empty lots predefined on the map as long as they do not already have a tower built on them. Constructing a tower costs gold, similarly player can sell an already constructed tower which turns the area back to an empty lot but also reimburses a portion of the gold spent to construct that tower back to the player.

Each tower picks its target as “the enemy who progressed farthest on the path” out of those available to that tower limited by its range (radius in which that tower can reach an enemy to attack). The towers can not attack out of reach enemies, and the enemy who progressed farthest among reachable enemies would be the one closest to the exit in terms of distance on the path and not the Euclidean distance to exit point (i.e. distance of a straight line which doesn’t take the path into account, like a birds flight path).

The movement of enemy units will be continuous over time and in terms of pixels depending on their speed; it will **not** be block by block according to the tile grid of the map. The same goes for the projectiles fired form the towers.

3. Game view structure:

3.1 Main screen

The game starts with the main screen, which provides the options to start a new game, open the level editor, open the options page or to quit the game; feel free to customize the main screen to your liking and aesthetic of your game.

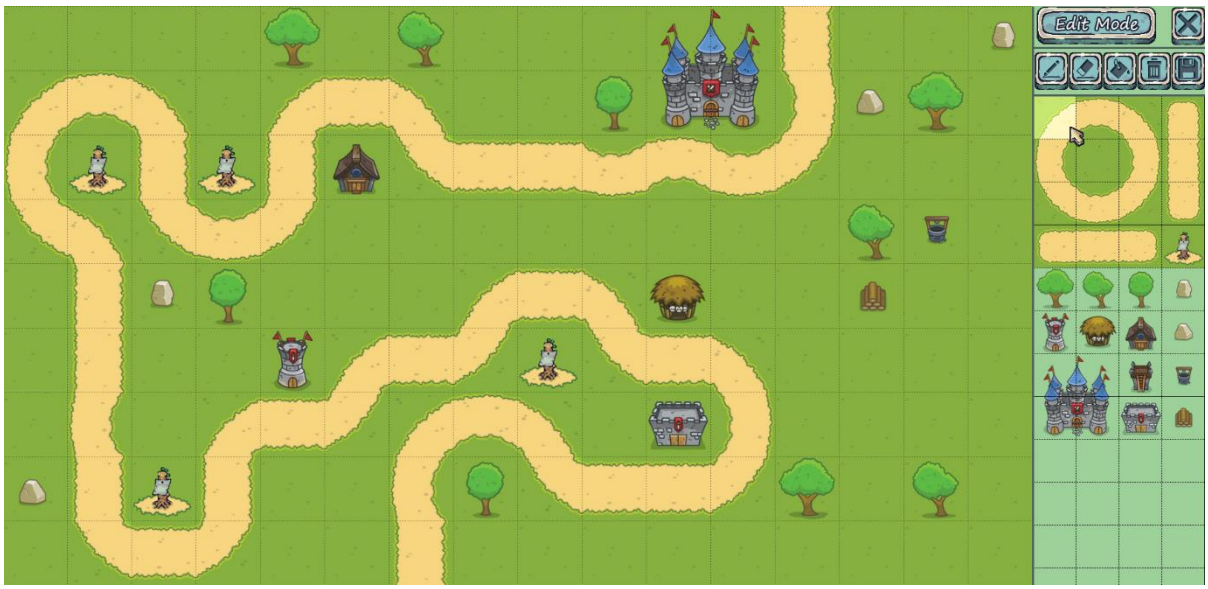
3.2 Options screen

Option screen should provide the player with some global options like the following:

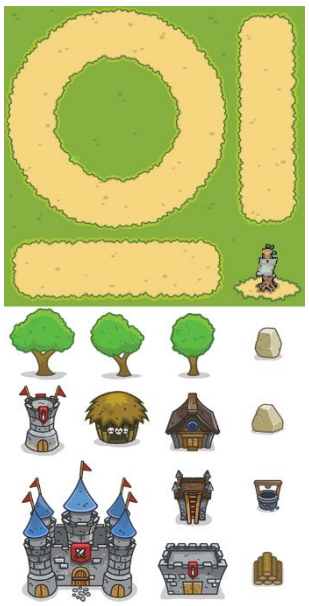
- **number of waves** in total, the **number of groups** per wave, the **number of enemies** per group of a certain wave,
- the **delay** between the waves, the delay between the groups of a wave, the delay between enemies of a group,
- the **composition** of types of enemies for a given group or a wave in terms of number of goblins and number of knights.
- the starting **amount of gold** for player and the amount of gold you earn when you defeat a goblin or a knight
- the **starting hit point** of the player, the **hit points** of a goblin and a knight, the **damage dealt** (in terms of hit points) by an arrow, an artillery shell, and a magic spell.
- the **construction cost** of each type of tower
- the effective **range** (as radius) and **rate of fire** for each type of tower, the range of AOE damage for the artillery shells
- the **movement speed** of a goblin and of a knight

You should initialize these parameters with **sensible defaults** and provide a button to revert the options back to those defaults. The game should retain the values set for options across all the following game sessions even if the player closes the game application.

3.2 Map Editor



The map editor should allow the player design a custom map by placing tiles from a given set of tiles and save it as a file. The map editor should work as a tile based editor and let the user place the tiles by first clicking to select a tile from the list of available tiles and then clicking to grid cells representing the map (optionally you can support darg & drop placement of tiles). The editor should provide tiles for following functionalities:



- tiles to build the main path the enemies will be following through (it does **not** have to support branching paths)
- tiles to place a pre-constructed tower
- a tile for the empty lot where a tower can be constructed during game play
- a tile for the empty space (i.e. just grass)
- some decorative tiles with no functionality (i.e. trees, stones, objects, etc..)

The path should start and end on tiles at the edge of the map. The map editor should provide a way to let the player mark the beginning of the path and the end point of the path. The map editor should check the following conditions when the user tries to save the map as a file and should show a warning if they are not met:

- The starting tile of the path is at the edge of the screen and marked as starting point
- The ending tile of the path is at the edge of the screen and marked as ending point
- The path is fully connected and does not contain discontinuities (i.e. exit is reachable from the starting tile)
- There are at least four tiles with empty lots on which towers can be built during play

Feel free to use artistic liberty with the graphical assets, overall design and UI as long as it includes at least the same functionality; also feel free to be creative with the objects and enemies. In other words, you can use objects other than the ones shown above or provided in the graphical assets package and you can implement enemies other than goblins and knights.

3.3 Play Mode:

When player click new game on the main menu the game would show the **list of saved maps**, upon selection of a map the game proceeds to play mode where the selected map is loaded and displayed. (*note: the sample screenshot above shows branching paths, but you are not required to implement these at this point*)

When the game play starts it should allow 4 seconds of a **grace period** before the enemies from the first group of the first wave start appearing on the start point of the path, to allow the player to construct his/her first towers.

The player can **construct new towers** on empty lots by clicking on them which brings forward a popup menu providing the options to build an archer tower, artillery tower or a mage tower. Following players selection if the player has enough gold covering the cost of the selected tower type, the game should replace the empty lot with selected tower and deduce the cost of construction from the amount of gold player owns. The player should be able to get out of the construction popup menu without making a selection by clicking any other part on the screen.

The game screen should provide a button for **pausing** the game and another button which toggles the game speed between the regular speed and **accelerated speed** to prevent long waits when there a little interaction between enemies and towers.

The game screen should also provide visual counters for the amount of **gold** the player has for the purpose of tower construction, the number of **lives** the player has and the number of **current wave** in progress side by side with the total number of waves (like 2/3 for second wave out of three).

Each enemy on the screen should have a **health bar** on top or bottom of it showing the remaining health represented with a colored bar. It should be updated according to remaining hit points after each hit received by the enemy.

When the mouse pointer moves over a tower a circle showing the boundaries of the attack **range of the tower** should be displayed on the screen and should remain visible as long as the mouse pointer is above that tower.

During the game play towers should attack if there is an enemy in range, each **projectile** fired should be displayed (arrow, artillery shell, spell) moving from the tower to its target.

Enemies should appear at the designated start point of the path according to the schedule setup on the options screen. When enemies appear on the screen they should start following the path according to the speed setup on the options for their type.

When an enemy reaches the end of the path at the edge of the screen undefeated it should disappear and one hit point should be deducted from the player. If the hit points of the player reach 0, the game should display a **Game Over banner** and return to main screen after a mouse click.

If an enemy is defeated before reaching the end of the path by getting hit by a projectile and having hit points less or equal to 0, then the enemy simply disappears from the screen, the amount of gold owned by the player is increased by the amount earned from the defeat of that type of enemy as set on the options screen.

4. Technical Specifications:

You are expected to code your application with Java.

You are allowed to use Swing or JavaFX but **no** other graphics libraries, game libraries or game engines[2]. You are free to pick any one of the Swing and JavaFX, but let us remind you

[2] examples for libraries **not** allowed: libGDX, LWJGL, and jMonkey engine

that JavaFX is the successor of Swing (*it is designed to replace Swing*), it is more modern in terms of programming and benefits from hardware acceleration, still first investigate code examples for both and make a team decision on which one to use. If you are in doubt whether a certain library is permitted for use, please contact your TA.

For writing structured data to disk you have two options, you can either:

- use Java's Serializable interface and native serialization classes, or
- you can implement your own code to write data to a file from scratch using FileReader / FileWriter or FileInputStream / FileOutputStream.

Even though this design document over-specifies a lot of aspects of the project, still some design aspects, technical details and creative options are left under-specified on purpose; you are expected to discover those as you progress and guide your project with your team's own design decisions.

5. Credits:

Give proper credit on your game (by including a credits.txt in your git repo) if you use any of the graphics from the provided asset package. (see credits.txt in the assets zip file)

Resources:

- Using Object Serialization through **java.io.Serializable** interface and **java.io.ObjectInputStream** / **java.io.ObjectOutputStream** classes make your life easier on this type of object persistence tasks. See:
  - https://en.wikipedia.org/wiki/Serialization#Java
  - https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html
  - https://docs.oracle.com/javase/8/docs/api/java/io/ObjectInputStream.html
  - https://docs.oracle.com/javase/8/docs/api/java/io/ObjectOutputStream.html
  - Some examples:
    - https://www.geeksforgeeks.org/serialization-in-java/
    - https://www.digitalocean.com/community/tutorials/serialization-in-java

- Resources for **JavaFX**:
  - https://openjfx.io/
  - https://en.wikipedia.org/wiki/JavaFX
  - https://wiki.openjdk.org/display/OpenJFX
  - https://docs.oracle.com/javase/8/javafx/get-started-tutorial/title.htm