# Towers

Assignment 3
CSSE1001/7030
Semester 1, 2018

Version 1.0.0
20 marks

Due Friday 1st June, 2017, 18:00

# 1. Introduction

This assignment provides you the opportunity to apply concepts taught throughout the course to extend the functionality of a basic tower defence game.

The assignment will focus on the concept of Graphical User Interfaces (GUIs) and object-oriented programming. You will be required to extend the functionality of the base game to achieve marks.

Students are encouraged to review some tower defence games, to better understand how this type of game is played and for inspiration on advanced features.

Because this assignment deals with multiple files, while not required, you may wish to investigate a more sophisticated IDE. One option is PyCharm, which is free for students.

*In the first week after this assignment is released, some updates are expected to be made to improve the quality of the support code. These will be non-breaking changes to the assignment code and will not be to the* `a3.py` *file. You can incorporate the changes into your assignment by overwriting the modified files in the project.*

## 1.1. Game Play

*Shortly after this assignment's release, a link to a video overview will be added here.*

# 2. Overview

## 2.1. Getting Started

The archive `a3_files.zip` contains all the necessary files to start this assignment. A significant amount of support code has been supplied to make it possible to begin with a simple application that is almost working.

The main assignment file is `a3.py`, which contains an incomplete implementation of `TowerGameApp`, the top-level GUI application class. The other files are support code which **must not** be edited. Initially, you do not need to understand much of this code, but as you progress through the tasks, you will need to understand more of this code. You should add code to `a3.py` and modify `TowerGameApp` to implement the necessary functionality.

*You are permitted to create additional files to simplify the separation of tasks (i.e. task1.py, task2.py, etc.), although this is not required. If you do this, `a3.py` must be the entry point to your application. One way to achieve this is to move `TowerGameApp` to a separate file, such as `base.py`. **Regardless of how you structure your files, the code must all be able to be demonstrated by running `a3.py`.***

# 3. Assignment Tasks

## 3.1. Task Overview

This assignment is broken down into three main tasks:

1. The first task involves adding lines of code to clearly marked sections within the main assignment file
2. The second task involves extending the design to add more interesting functionality to the game
3. And the third task involves adding sophisticated functionality to further improve the gameplay experience

**For post-graduate students only**, there is an extra task that involves doing independent research.

In general, as the tasks progress, they are less clearly prescribed and increase in difficulty.

## 3.2. Task Breakdown

CSSE1001 students will be marked out of 20 & CSSE7030 students will be marked out of 26 based on the following breakdown. Tasks may be attempted in any order, but it is recommended to follow this breakdown, top-down, completing as much as possible of each task before moving on to the next.

| | Sub–Task | Marks |
|---|---|---|
| | | **9 marks** |
| | App Class | 1 marks |

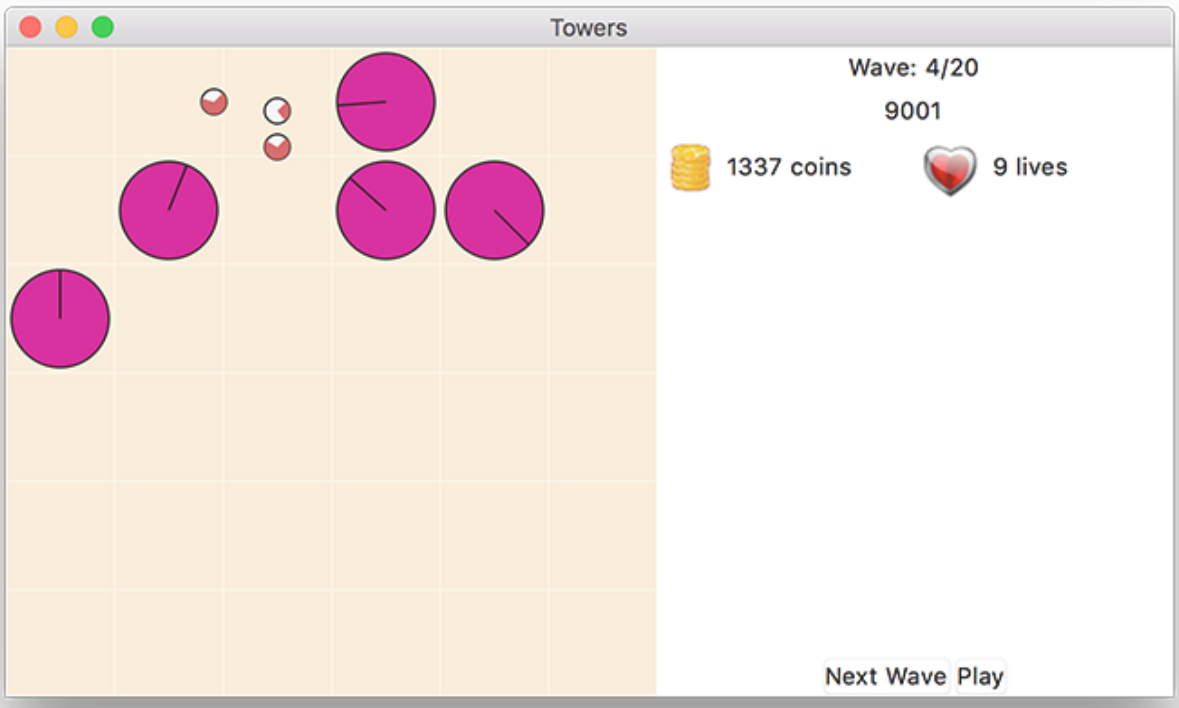
| | Sub–Task | Marks |
|---|---|---|
| **Task 1**<br>*Basic Features* | Tower Placement | 3 marks |
| | StatusBar Class | 2 marks |
| | File Menu & Dialogs | 1 marks |
| | Play Controls | 2 marks |
| | | **7 marks** |
| **Task 2**<br>*Intermediate Features* | Sell Tower | 1 mark |
| | Custom Tower & Enemy | 2 marks |
| | Shop Class | 3 marks |
| | High Scores | 1 marks |
| | | **4 marks** |
| **Task 3**<br>*Advanced Features* | Advanced Tower | 1.25 marks |
| | Advanced Enemy | 1.25 marks |
| | Upgrade Tower | 1.5 marks |
| | | **6 marks** |
| **Post-Graduate Task**<br>*Independent Research* | Selection of Library or Technique | 2 marks |
| | Application of Library or Technique | 2 marks |
| | Integration into Existing Game | 2 marks |

## 3.3. Mark Breakdown

For each task, marks will scaled according to the following breakdown.

| | Description | Marks |
|---|---|---|
| **Code Quality** | Code is readable. Appropriate and meaningful identifier names have been used. Simple and clear code structure. Repeated code has been avoided. | **15%** |
| | Code has been simplified where appropriate and is not overly convoluted. | **10%** |
| | Documented clearly and concisely, without excessive or extraneous comments. | **15%** |
| **Functionality** | Components are functional, without major bugs or unhandled exceptions.<br>*Assessed through user testing/playing, not automated testing.* | **60%** |

## 4. Task 1 – Basic GUI




Basic GUI Example

There are a significant number of comments in `a3.py` intended to help you complete this task.

## 4.1. App Class

Write a `main` function that launches the `TowerGameApp` GUI. Call this `main` function inside an `if __name__ == ...` block.

Modify `TowerGameApp` so that the title of the window is set to something appropriate (i.e. Towers).

## 4.2. Tower Placement

Allow the user to place a tower by clicking a grid cell with the mouse. While their mouse is being moved over the `GameView`, show a preview of the tower and its range in the current cell as well as the new path the enemies would take if a tower were placed there. When the mouse leaves the `GameView`, hide the preview.

If placing a tower in the current grid cell would make it impossible for the enemies to move through the grid, or there is already a tower there, the preview should indicate that this would not be a legal placement by showing an X shape instead of the tower.

See `GameView.draw_preview`, `GameView.draw_path`, `GameView.draw_towers`, `TowerGame.place`

## 4.3. StatusBar Class

Define a class named `StatusBar`, which inherits from `tk.Frame`. This class is used to display information to the user about their status in the game. The `StatusBar`'s widgets must:

1. be updated whenever necessary (i.e. after an enemy dies, etc. - see `Task 1.3 (Status Bar): Update ...` comments)
2. be laid out *approximately* according to Basic GUI Example
3. contain the following widgets:
    - **Wave (first row; centre)** A label to display the current wave
    - **Score (second row; centre)** A label to display the player's score
    - **Gold (third row; left)** A label to display the amount of coins the player has in their wallet, with an image of gold coins to the right
    - **Lives Remaining (third row; left)** A label to display the number of lives the user has remaining, with an image of a heart to the right

**Note:** For convenience, you should have a setter method for each of the relevant widgets. i.e. `set_score(score)`, etc.

The `StatusBar` class should be added to the application in a frame to the right of the `GameView`. Other widgets will be added to this frame in subsequent tasks.

### 4.4. File Menu & Dialogs

Implement a menu bar, with a `File` menu. The File menu should have the following entries:

- `New Game`: Restarts the game
- `Exit`: Exits the application

When the user attempts to exit the application, either by the file menu or otherwise, they should first be prompted with a dialog to confirm that they indeed want to quit the application. Further, when the game is over, the user should be shown a dialog informing them of the outcome (either win or loss).

**Note:** On Mac OS X, the file menu should appear in the global menu bar (top of the screen).

### 4.5. Play Controls

Add a frame to the right of the `GameView`, below the `StatusBar`. Then, add a button to the frame to send another wave of enemies against the player. Lastly, add a second button to the frame to pause/resume the current wave. Both buttons should be disabled when the game is over and (re)enabled when a new game is started.

## 5. Task 2 – Intermediate Features

### 5.1. Sell Tower

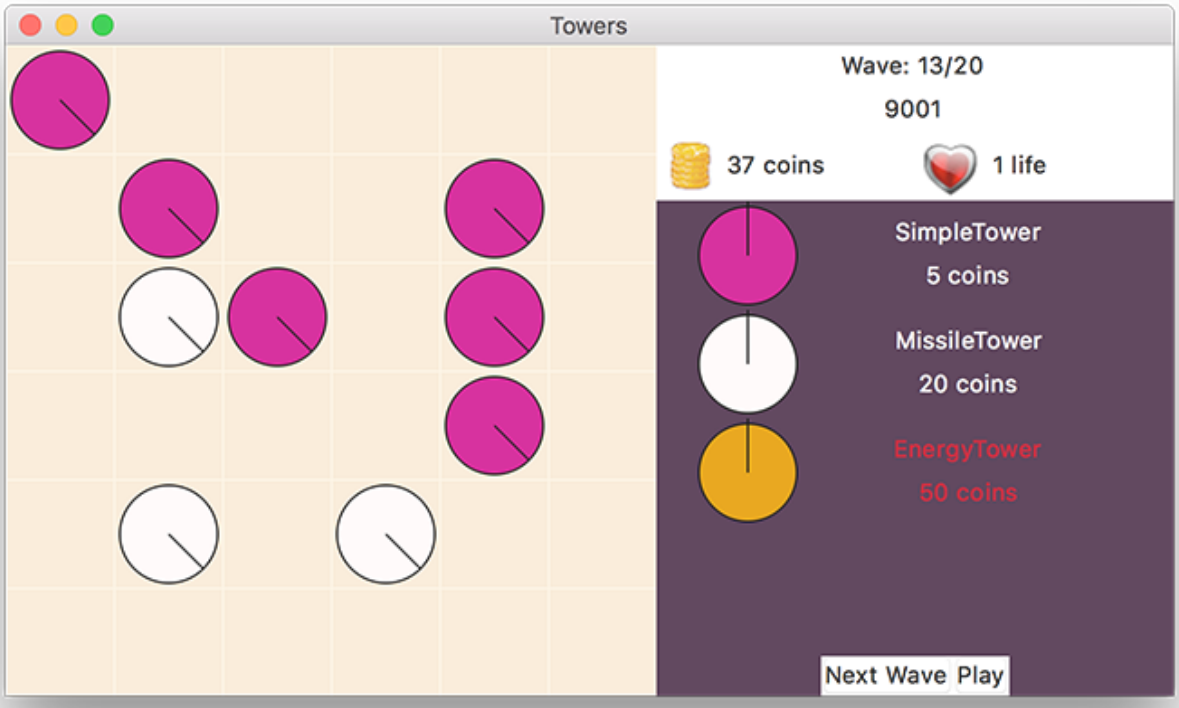
Add a feature that allows the player to sell a tower by right clicking on it. The tower should be removed from the grid and 80% of its value should be put back into the player's wallet.

See `TowerGame.remove`

### 5.2. Custom Tower & Enemy

Implement a subclass of `AbstractEnemy` that is immune to `projectile` & `explosive` damage. Next, implement a subclass of `AbstractTower` that deals `energy` damage (for use against your custom enemy).

### 5.3. ShopView Class




Shop View

The shop allows the user to select other types of towers to use against the enemies.

The shop must display each of the available towers, alongside with their name & cost in gold (as shown above). The gold cost should be displayed in red if the user is unable to afford the tower (this should be updated whenever the user gains or spends gold).

Add the shop to the screen, to the right of `GameView`, below `StatusBar` & above the play controls. Add `SimpleTower`, `MissileTower`, & your custom tower from 5.2. Custom Tower & Enemy to the game.

An incomplete implementaiton of the `ShopView` class has been provided, with an example of how to instantiate it and add it to the game. This task involves implementing the `ShopTowerView` GUI class required to display an individual tower in the shop, and integrating it into `ShopView`.

The `ShopTowerView` class must inherit from `tk.Frame` and have at least the following methods:

- **__init__:** Must have at least the following arguments:
  - tower: The tower to display (an instance of AbstractTower)
  - click_command: A callback function to be called when the tower is clicked

    - *args & **kwargs: Positional & keyword arguments to be passed to the `tk.Frame constructor`
- **set_available(**enabled: bool): Updates the widget to show whether the tower is available (white text) or not (red text)

### 5.4. High Scores

Add an item to the file menu called*HighScores*. When clicked, this should display a new window which displays the names and scores of the top ten scoring players. When the user runs out of lives, they should be prompted for their name if their score is enough to get in the high scores list.

**Note:** The `HighScoreManager` class will be useful for managing high scores.

## 6. Task 3 – Advanced Features

### 6.1. Advanced Tower

Create a more sophisticated tower & add it to the shop. This tower must make the game substantially more interesting by offering functionality that differs from the provided towers. Possible options for a new tower could be a flame throwing tower, pulse tower, boost tower, slow tower, ...

A key determination in marking this advanced feature is that the new tower requires non-trivial algorithmic logic to implement its functionality.

### 6.2. Advanced Enemy

Create a more sophisticated enemy & add it to the game. This enemy must make the game substantially more interesting by offering functionality that differs from the provided enemies. Possible options for a new enemy could be one that shoots at towers, dealing damage to them, or an enemy that spawns new enemies, ...

A key determination in marking this advanced feature is that the new enemy requires non-trivial algorithmic logic to implement its functionality.

### 6.3. Upgrade Tower

Allow players to upgrade a tower. If a player left-clicks on a tower in the grid allow them to select an upgrade option for the tower. This will activate a set of checkbox options in a control underneath the play controls. Each option will have an associated cost. Selecting an option will deduct the cost from the player's wallet and add that feature to the tower.

## 7. CSSE7030 Task – Independent Research

### 7.1. Advanced Feature

Research a library or design technique that has not been covered in CSSE7030. Use this to implement a significant new feature in the game. For example: you could add sound effects to the game; you could create multiple levels of game play, with additional challenge on each level; You could use a design pattern to facilitate new functionality within the game (e.g. towers aging and losing effectiveness).

## 8. Assignment Submission

**Note:** There will not be a practical interview for the third assignment.

Your assignment must be submitted via the assignment three submission link on Blackboard. You must submit a zip file, `a3.zip`, containing `a3.py` and all the files required to run your application (including images). You may omit the support code — if you do so, the most recent version will be used.

Late submission of the assignment will **not** be accepted. Do not wait until the last minute to submit your assignment, as the time to upload it may make it late. Multiple submissions are allowed, so ensure that you have submitted an almost complete version of the assignment well before the submission deadline of 6pm. Your latest, on time, submission will be marked. Ensure that you submit the correct version of your assignment. In the event of exceptional circumstances, you may submit a request for an extension. See the course profile for details of how to apply for an extension.

Requests for extensions must be made **no later** than 48 hours prior to the submission deadline. The expectation is that with less than 48 hours before an assignment is due it should be substantially completed and submittable. Applications for extension, and any supporting documentation (e.g. medical certificate), must be submitted via my.UQ. You must retain the original documentation for a *minimum period* of six months to provide as verification should you be requested to do so.

## Change Log

Any changes to this document will be listed here.