Team FooBar
API Design Document

GAME GENRE: Tower Defense Game

GENRE DESCRIPTION/QUALITIES

The goal of a Tower Defense game is to survive as many waves of attack that you can. A user loses when they lose all of their lives (enough enemies reach the end of the path). In every level, the enemies are stronger, and it becomes harder and harder for the tower to survive. A Tower Defense game requires towers, paths, treasures, enemies, spawn rates, and cheat codes.

GAME DESIGN GOALS

- Reading from a user-configured text to initialize the game, using reflection to create objects dynamically
- Decoupling the Model and View by using a Controller. The Model pushes updates to the Controller and the View pulls data from the Controller.
- Having flexible/configurable user interface: The user will be able to choose basic game info, such as game name, splash image, number of levels, etc., as well as defining a map path, types of enemies, types of towers, and individual level specifications.

PRIMARY CLASSES/METHODS

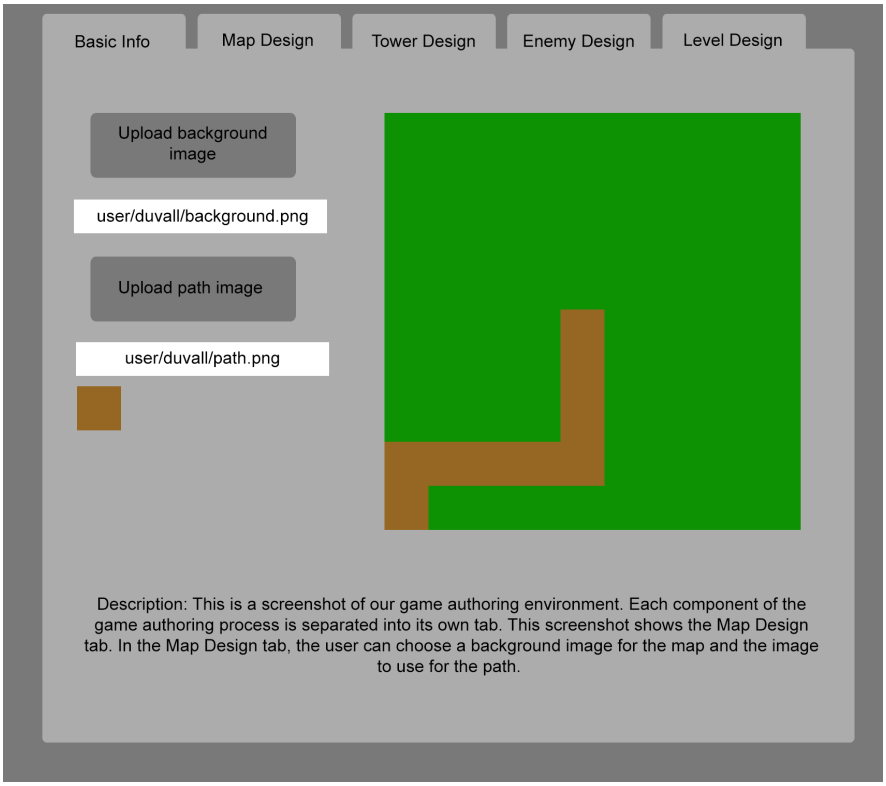
---

1. Game Authoring Module

```
class GameAuthoringGUI
class Tab
class BasicInfoTab extends Tab
class MapDesignTab extends Tab
class TowerDesignTab extends Tab
class EnemyDesignTab extends Tab
class LevelDesignTab extends Tab

class BasicInfoData {
      public String getGameName()
      public int getGold()
      public int getLives()
      public int getNumLevels()
      public String getSplashImage()
      public void setGameName(String gameName)
      public void setGold(int gold)
      public void setLives(int lives)
      public void setNumLevels(int numLevels)
      public void setSplashImage(String splashImage)
```

```
}

class TowerDesignData {
	public void addTowerAttackSpeed(int towerAttackSpeed)
	public void addTowerCost(int towerCost)
	public void addTowerDamage(int towerDamage)
	public void addTowerID(String towerID)
	public void addTowerImage(String towerImage)
	public void addTowerLife(int towerLife)
	public void addTowerRecyclePrice(int towerRecyclePrice)
	public List<Integer> getTowerAttackSpeeds()
	public List<Integer> getTowerCosts()
	public List<Integer> getTowerDamages()
	public List<String> getTowerIDs()
	public List<String> getTowerImages()
	public List<Integer> getTowerLives()
	public List<Integer> getTowerRecyclePrices()
}

class MapDesignData {
	public boolean[][] getMap()
	public String getMapBGImage()
	public String getPathImage()
	public void initializeMap(int width, int height)
	public void setMapBGImage(String mapBGImage)
	public void setMapData(int x, int y, int isPath)
	public void setPathImage(String pathImage)
}

class EnemyDesignData {
	public addEnemyDamage(int enemyDamage)
	public addEnemyGoldValue(int enemyGoldValue)
	public addEnemyImage(String enemyImage)
	public addEnemyLives(int enemyLife)
	public addEnemySpeed(int enemySpeed)
	public List<Integer> getEnemyDamages()
	public List<Integer> getEnemyGoldValues()
	public List<String> getEnemyImages()
	public List<Integer> getEnemyLives()
	public List<Integer> getEnemySpeeds()
}

class LevelDesignData {
	public void addEnemyQuantity(HashMap<String, Integer> enemyData)
	public List<HashMap<String, Integer>> getEnemyQuantity()
}
```

Game Authoring User Interface:



2. Game Engine (Backend) Module

```
//Backend, fill in stuff here
```

3. Game Engine (Frontend) Module

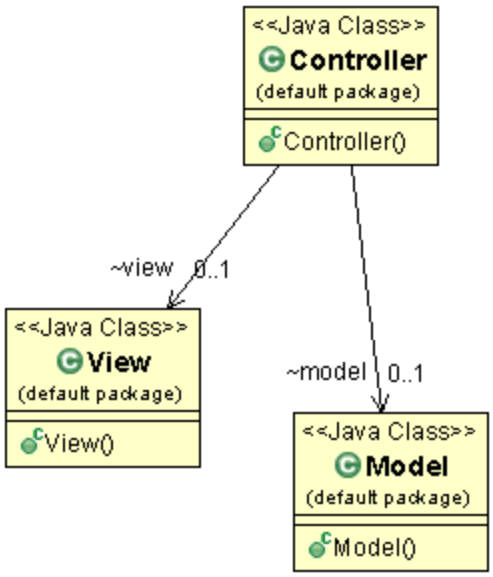
```
class Game extends JGEngine{
      //jgame instance of the game
      public void initializeGame(Initializer initializer);
}

class CanvasPanel extends Panel {
      //Panel to hold the jgame instance
}

class GameMenu extends Menu {
      //Menu to hold file...options
}

class StatsPanel extends Panel {
      //Panel to display tower stats
      public void updateStats(Tower tower);
}
```

```
class StorePanel extends Panel {
      //Panel to allow users to purchase towers
      bool itemSelected;
}

class View extends Panel {
      //Main view class, facilitates communication with the controller
      Controller controller;
      String jsonURL;
      public void itemBought(int[] position, Tower tower);
      public String getJsonURL();
}
```

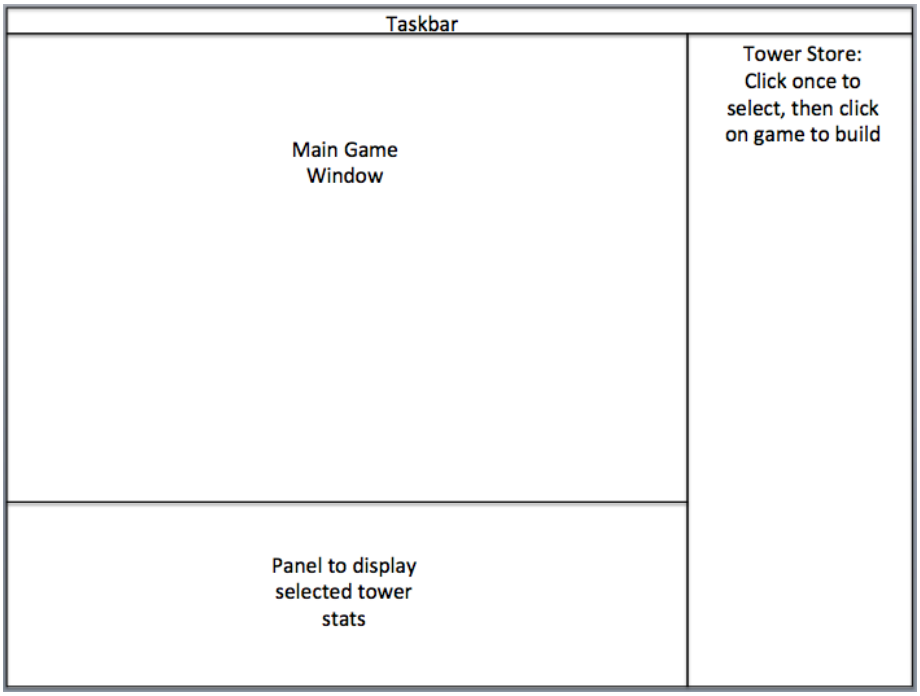
4. Communication Between Game Engine Frontend and Backend

```
class Controller{
      Model model;
      View view;
      public void newGame(File jsonFile);
      public void setMoney(int money);
      public Initializer initializer;
      public int getMoney();
      public int getLife();
      public void setTower(Type type, Position pos);
      public List<Tower> getTowers();
      public List<Enemy> getEnemies();
      public List<Bullet> getBullets();
      public List<Path> getPath();

      public Image getPathImage();
      public Image getBackgroundImage();
      public Initializer getInitializer();
      public Tower getTowerStats();
}

class Initializer{
      // Holds all the variables needed to initialize the game
      int money;
      int lives;
      String backgroundURL;
      List<Integer> path;
      List<Tower> towers;
}
```

Game Engine Backend/Frontend UML:



Game Engine Frontend User Interface:



EXAMPLE CODE

The data file would have lots of data fields that initialize the environment. Some of these data fields would be the background image location, the name of the game, the number of lives that a user has, and the number of levels.

Example game: Bloons Tower Defense

```
{
        "name": "Bloons Tower Defense",
        "splashImage":
        "http://4.bp.blogspot.com/-T5XPeO62Y7g/TveD6az87qI/AAAAAAAADNc/KNrOg5EtHGw/s1600/bloonsTD5.JPG",
        "gold": 650,
        "numberOfLives": 3,
        "levels": 50,
        "difficultyScale": 1

}
```

Example game: Steampunk Tower Defense

```
{
        "name": "Steampunk Tower Defense",
        "splashImage":
        "http://cache.hackedfreegames.com/uploads/games/pictures/022/pH8ZS4YZY3HGU.jpg"
        "gold": 11000,
        "numberOfLives": 5,
        "levels": 100,
        "difficultScale": 1
}
```

DESIGN ALTERNATIVES

The main challenge in our design was to devise an effective way to break down the tasks between the Frontend and Backend in such a way that each team could work on their components independently and communicate solely through the controller. Another alternative that we considered was for both teams to work on the JGame classes together, and only separate the teams by the tasks that each would perform. However, we ultimately decided that this alternative would be messier from a design perspective, and could lead to a number of issues where the division between the Frontend and the Backend became very vague.

TEAM RESPONSIBILITIES

Game Developer Environment

- Susan Zhang
- Rebecca Lai

Game Engine Frontend

- Lalita Maraj
- Alex Zhu

Game Engine Backend

- Wenxin Shi
- Fabio Berger
- Yuhua Mai
- Harris Osserman
- Jiaran Hao
-