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.
- Decoupling the parts of the View to allow them to operate independent of each other.
- 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

package gameAuthoring

```
class BasicInfoTab {
      public BasicInfoTab(GameData)
      public MouseAdapter setInfoListener()
      public MouseAdapter setSplashImageListener()
}

class EnemyDesignPanel {
      public EnemyDesignPanel(EnemyDesignTab)
      public MouseAdapter createEnemyButtonListener(EnemyDesignPanel)
      public MouseAdapter createPathListener()
}

class EnemyDesignTab {
      public EnemyDesignTab(GameData)
      public void addEnemy(File, String)
}
```

```
class GameData {
	private void addDataToContainer()
	public void addEnemy(String, int, String, int, int)
	public void addLevel(LevelJSONObject)
	public void addTower(String, String, int, int, int, int, int)
	public JSONArray getEnemyList()
	public void setDifficultyScale(float)
	public void setGameName(String)
	public void setGold(int)
	public void setLives(int)
	public void setMap(String, String, Point2D, Point2D,
Collection<Point2D>)
	public void setSplashImage(String)
	public void setTilesPerRow(int)
	public void setWindowHeight(int)
	public void setWindowWidth(int)
	public void writeToFile()
}

class Grid {
	public Grid(int, int)
	public void addCoordinate(Point2D)
	public MouseAdapter addPathDoneListener()
	public Collection<Point2D> getPathCoordinates()
	public Point2D getPathEnd()
	public Point2D getPathStart()
	public boolean isValidPath(int, int, int, int)
	public boolean isValidPathHelper()
	public void paintComponent(Graphics)
	public void removeCoordinate(Point2D)
	public void setBackgroundImageSource(File)
	public void setImageSource(File)
	public void PathEnd(Point2D)
	public void setPathStart(Point2D)
}

class GridButton {
	public void addPathListener(GridButton)
	public Point2D getCoordinate()
	public boolean isPath()
	public void setImageSource(File)
}

class LevelDesignDialog {
	public LevelDesignDialog(LevelDesignTab, LevelDesignPanel)
	public MouseAdapter createWaveDesignListener(LevelDesignDialog)
}
```

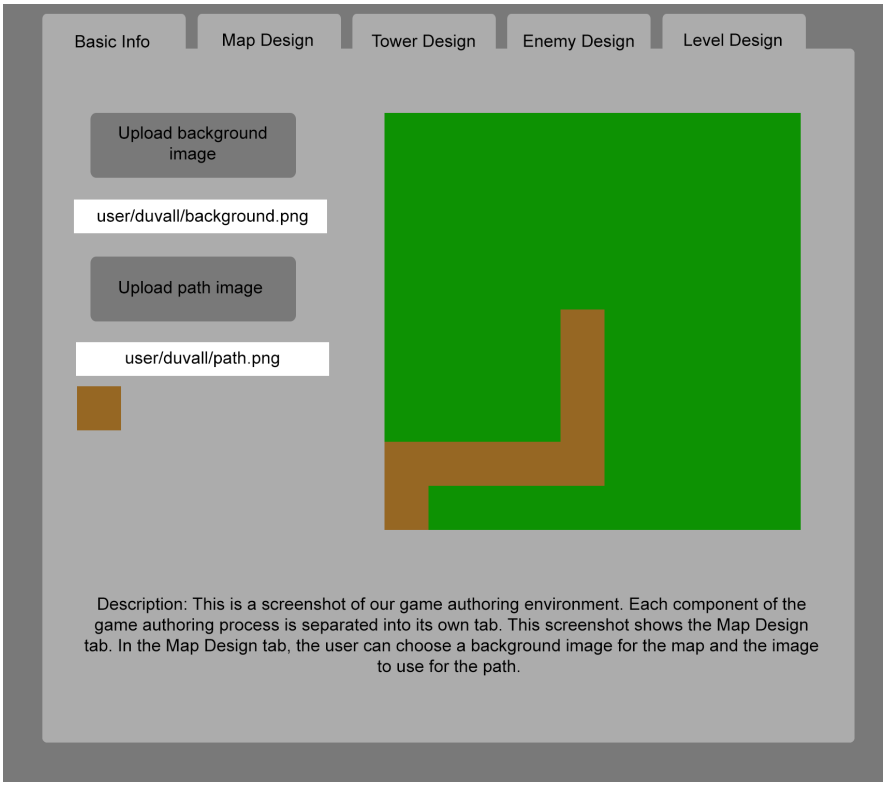
```
class LevelDesignTab extends Tab {
      public LevelDesignTab(GameData)
      public void addLevel(LevelJSONObject)
      public int getLevel()
}

class MapDesignTab extends Tab {
      public MapDesignTab(GameData)
      public MouseAdapter createGridBackgroundListener(Grid)
      public MouseAdapter createPathCheckListener()
      public MouseAdapter createPathListener()
}

class MiscellaneousTab extends Tab {
      public MiscellaneousTab(GameData)
      public MOuseAdapter createImageSelectionListener(JLabel)
}

class Tab {
      public Tab(GameData)
      public GameData getGameData()
      public JPanel getTab()
}

class TowerDesignTab {
      public TowerDesignPanel(TowerDesignTab)
      public MouseAdapter createPathListener()
      public MouseAdapter createTowerButtonListener(TowerDesignPanel)
}

class EnemyDesignTab {
      public void addEnemy(File, String)
}

class EnemyDesignPanel {
      public MouseAdapter createEnemyButtonListener(EnemyDesignPanel)
      public MouseAdapter createPathListener()
}

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

package gameAUthoring.JSONObjects

```
class EnemyJSONObject {
```

```
      public EnemyJsonObject(String, int, String, int, int)
}

class LevelJSONObject {
      public LevelJSONObject(int, int, List<Map <String, Integer>>)
}

class MapJSONObject {
      public MapJSONObject(String, String, Point2D, Point2D,
Collection<Point2D>)
}

class PointJSONObject {
      public PointJSONObject(Point2D)
}

class TowerJSONObject {
      public TowerJSONObject(String, String, int, int, int, int, int)
}

class WaveJSONObject {
      public WaveJSONObject(int, Map<String, Integer>)
}
```

Game Authoring User Interface:



2. Game Engine (Backend) Module

```
public interface TowerInfo {
//all tower and tower factory
//* implements TowerInfo so that front end people can
//* get tower info properly without exposing tower and
//* towerfactory.
    public String getTowerName ();
    public double getX ();
    public double getY ();
    public double getDamage ();
    public double getAttackSpeed ();
    public int getAttackMode();
    public double getRange ();
    public int getCost ();

    public double getRecyclePrice ();
    public String getDescription();
    public String getImage ();
}

class Tower extends JGObject{
      double damage;
      double attackSpeed;
```

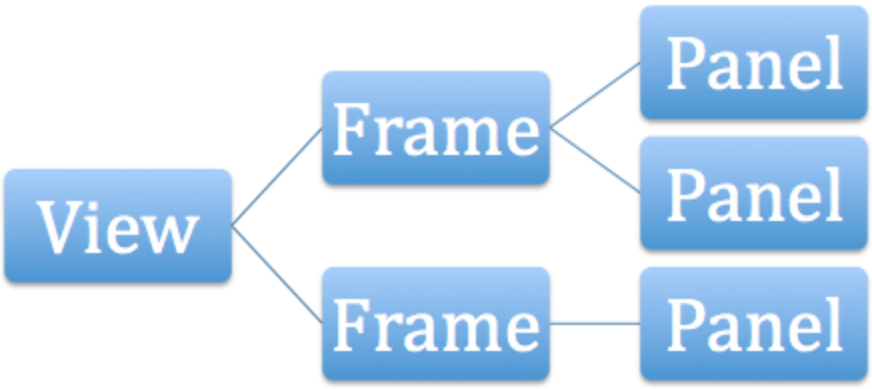
```
	double range;
	double cost;
	double recyclePrice;
}

class Enemy extends JGObject{
}

class Bullet extends JGobject{
	Enemy targetEnemy;
}

class GameInfo{
	//Store game information such as life, gold
}

class Grid{
	public void setOnPath(String imgPath)
	public void setTower()
	public boolean hasPath()
	public boolean hasTower()
	public boolean isEmpty()
	public double getX()
	public double getY()
	public double getEndX()
	public double getEndY()
	public double getCenterX()
	public double getCenterY()
	public String getPathImage()
}

class Rule{
	//Rule to control how each wave of enemy will behave
}

class Model {
	public Tile getTile(int x, int y)
	public LinkedList<Tile> getPathList()

}
```

3. Game Engine (Frontend) Module

The Frontend of the Game Engine team is known as the View. The controller (described below) communicates with the frontend via the View class. We use the View class as the core of all components of the View. The diagram below illustrates the relationship between the View and its Swing subcomponents. Rather than have the different parts of the View have access to the

controller, we gave them access to the View class. Therefore, the View serves as the view component's interface to the controller. This allows the other classes in the view to be completely decoupled from the controller, such that if there are changes in the controller, we would only have to make the appropriate changes in the View class.

```
class View extends Panel {
      //Main view class, facilitates communication with the controller
    private InitializationFrame initializationFrame;
    private GameFrame gameFrame;

    public void selectNewGame () {
       // displays the initialization frame so users can select a game
    }

    public void loadNewGame () {
       // facilitates the sequence of events to load a game
    }

    public void startGame () {
       // facilitates the sequence of events to start a  game
    }

    public void newGame (File file) {
        // facilitates the sequence of events to initiate a new game
    }


    public boolean buyTower (int x, int y, String tower) {
       // Asks controller if a user can buy the tower by placing it at given
      //x and y coordinates
      //returns true if purchase is valid
      // false otherwise; example: user tries to place tower directly on path
    }

/*This class would also contains a number of get methods that components
within the View execute in order to retrieve information from the controller.
Examples include getTowerInfo(), getLives() , getMoney(), etc.*/

}
```



The View has two interfaces to the user: an initialization Frame and a game Frame. In the initialization Frame, the user is able to select the JSON file that they would like to use to build their game. The gameFrame contains all of the panels that present the game experience to the user, such as the game display and the store.

Initialization Frame:

```
public class InitializationFrame extends Frame{

//displays the ItializationPanel that presents the user with File selection
//options
}

public class InitializationPanel extends Frame{

// takes the selected file and  sends it to the View class (to be sent to the
//controller)
}
```

GameFrame:

```
class Game extends JGEngine{
      //jgame instance of the game
      //uses the jgame doFrame() loop to update the other panels

      //monitors user actions within the game and calls the appropriate
      //methods to buy or select a tower etc.
      public checkUserInteractions();
}

class CanvasPanel extends Panel {
```

```
      //Panel to hold the jgame instance
      private Game game;
}


class StorePanel extends Panel {
      //Contains Panels that display available towers and whether a user can
      //purchase the towers
      //Contains Panels that display in depth information about each tower
      //Alerts the Game class that a user wants to place a tower via the
      //mediator

}
```

The Mediator:
The different parts of the gameFrame GUI need to communicate with each other occasionally. However, one of our design goals is to decouple the different GUI components. Thus, we use a Mediator class that facilitates communication between different parts of the gameFrame. Example methods are described below.

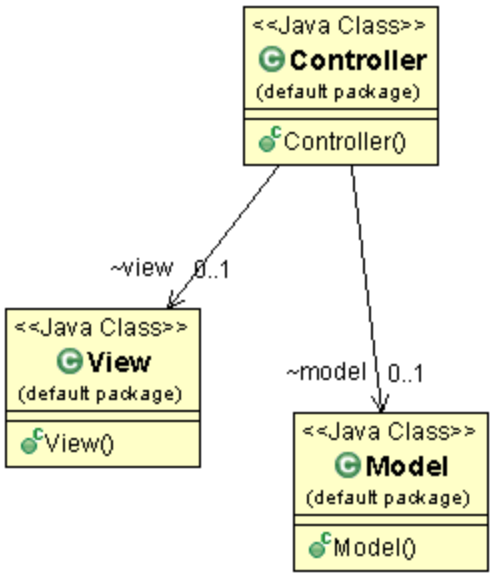
```
public class GameFrameMediator {




    public void endGame () {
      //Destroys the jgame instance so that it can be reloaded
    }


    public void placeTower (TowerInfo towerInfo) {
      //Notifies all colleagues that need to be updated
     // when a user is trying to purchase a tower
    }


    public void displayTowerInfo (TowerInfo tower) {
        // Used to display information about a tower
      //on the Tower Info Panel
    }


    public void purchaseTower () {
     // Executes the actions on the view components
    //that are impacted by the purchase of a tower
    }


    public void updateStoreStatus () {
```

```
        //Updates the enabled status of store items.
    }
}
```

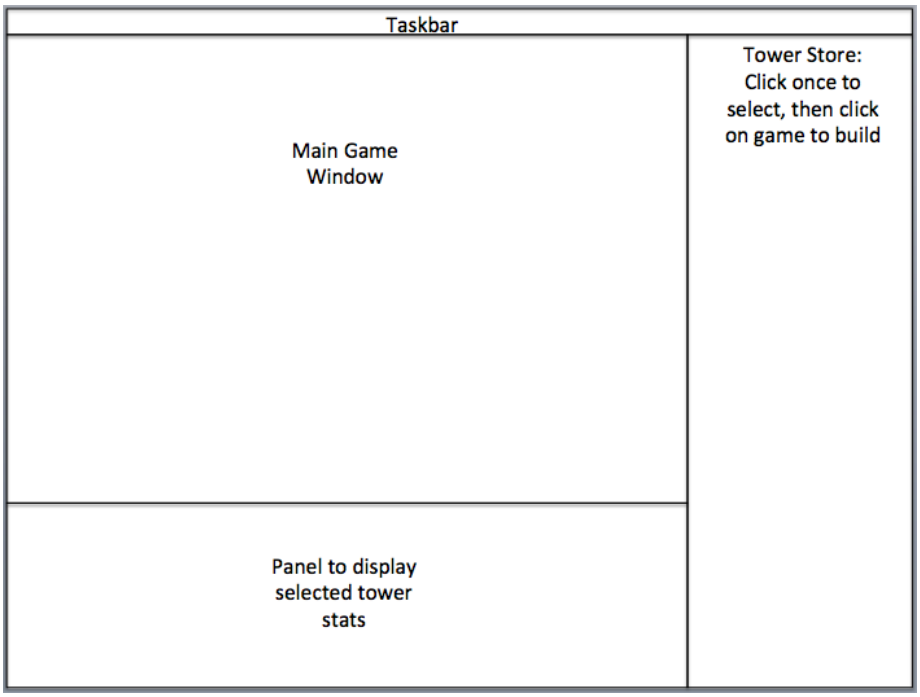
4. Communication Between Game Engine Frontend and Backend

```
class Controller{
      //Only acts on the View and Model not subcomponents of the View //and
      Model
      Model model;
      View view;

      //Passes the user selected JSON file to the model to start the
      //game
      public void newGame(File jsonFile);

      //Called by the view once the jgame instance is ready. Tells the
      //model to begin running the game.
      public void startGame();

      public int getMoney();
      public int getLives();

      //Sends user selected tower name and position to the model to
      //create a tower at the selected position
      public boolean purchaseTower (int x, int y, String name);
      public boolean sellTower(int x, int y);
      public boolean upgradeTower(int x, int y);
      public boolean setTowerAttackMode(int x, int y, int attackMode);

      public List<TowerInfo> getTowers();

      public List<PathInfo> getPath();

      public Image getPathImage();
      public Image getBackgroundImage();

}
```

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