VOOGASalad Application Program Interface (API)
Group: VOOGirls Generation

Authoring Environment

//Initializes the Program
public class **Main**
    public **start**(Stage s)
        Initializes the Initial Stage of the Program
    public static void **main**(String[] args)

//This is the GUI that is displayed to the user
public class **View**
    //list of Modules that were defined in the view and values they were given
    public **View**
        Initializes the various components in the View
    //These all initialize windows and menus for the GUI components
    public void **createGridEditor**()
    public void **createPieceEditor**()
    public void **createPatchEditor**()
    public void **loadWorkspace**(JSON file)
        Load previously created games using FileChooser to open for editing
    public void **createSaveButton**()
     // When pressed, call JSONwriter.save(List<Module>) and parse these into JSON format

//These are editors for various modifiable portions of the game - Grid, Piece, etc.
public class **GridEditor**
    public **GridEditor**()
        Initialize a grid of default size with buttons to customize
    public void **setSize**(int numRows, int numColumns)
        Change the view of the grid to represent this new size selected
    public void **setPatchShape**(String s)
        Sets the shape of the patches in the Grid
    public void **handle**(MouseEvent m)
        Create menu of all items in patch selected (including patch itself) for the user to choose one to edit

//EventCreator allows users to define complex rules (Events) by combining Triggers with //Actions
public class **EventCreator**
    public **EventCreator**
    public Event **makeEvent**(Trigger t, ActionList a)
        Maps an ActionList to a given Trigger. This allows the user to program complex behavior based on observables - when something happens (the Trigger), the game runs

the commands stored in the ActionList, e.g., Trigger: enemy loses all pieces.
ActionList: Victory message pops up, game restarts

public class **PieceCreator**
public **PieceCreator**
A collection of combo boxes and accordion menus representing options the user has to select from
public Piece **makePiece**(Modules m) ← overloaded method
Takes in a variable-sized list of Module objects. Module determines the behaviors that the unit has - e.g., movement, attacking, having an inventory, being able to swim, etc. Allows creation of custom units using *composition*

public class **PatchEditor**
public PatchEditor
Display options for customizing a patch

#GAMEDATA
public class **JSONWriter**
public **JSONWriter**
writes the current state of the game-in-progress to a JSON file so it can be loaded and worked on during another session.
public void **write**(List<Modules>)

Game Player

public class **GamePlayer**
The GUI of the game engine, initiates the scene, updates the game and sets
public **userInput**( )
returns the changes made by user interactions

public class **GameChooser**
public **chooseCurrentGame**( )
chooses the current game to run. Allows the user to switch or repeat games if needed.

//Keeps track of the high scores in the game
public class **ScoreManager**
public **ScoreManager**()
keeps track of the Scores of the player; loads and reads the game data
public List<Score> **getScores**()
return all the scores of the current players
public Score **getHighestScore**()

return the highest score on record

public class **GamePreferenceController**
public **setPreferences**(File properties)
sets the specific preferences for the current game played
public **saveSettings**()
saves the current progress in the game
public **setSettings**()
sets the game to the last saved settings of the game. Allows the user to use the previously saved settings when restarting the game.

Game Engine

public class **GameEngine**
public **GameEngine**
Initialize a GameLoop, JSONParser and prepares the engine for game loading

//Represents a Player of the Game. Can be hot swapped with human player control module //or AI implementation module through selection of subclasses
public class **Player**
public **Player()**

//Main GameLoop of the Game Engine.
public class **GameLoop**
public **GameLoop()**
public void **getInput()**
Called whenever a change happens in the game state (player makes a move/ behavior is executed)
public void **processInput()**
Executes rules and behaviors as a result of the state change and updates the state of the game with it's new state
public void **updatePieces()**
public void **updateGrid()**
public preference **getPreferences()**
Returns preference file containing preferences for the game
public void **setPreferences()**
Sets preferences for the game
public JSON **writeState()**
Writes the current state of the game to a JSON file
public void **setState()**
Sets the current state of the game from a JSON file

```
//Builds the state of a game from the games JSON file
public class GameBuilder
        public Game buildGame(JSON game)
                Utilizes the JSON properties file to create the appropriate instances of class
                using the ComponentFactory. Then combines the instances as defined by the JSON to
                build the state of the game in preparation for GameLoop running the game

//Uses Reflection to instantiate objects
public class ComponentFactory
        private Object createComponent(String s)
                Use reflection - may throw (ClassNotFoundException, NullPointerException)
        public Object getComponents()

//Takes a JSON and creates a list of objects using the Component Factory to parse the JSON
public class JSONParser
        public JSONParser
        public List<Object> parseJSON(JSONObject file)

//Represents a Piece
public class Piece
        public Piece()
        public Piece(Piece original)
        //cloning constructor. set all instance variables equal to the values from original
        public Piece(Map<JSON objects>)
        //import map of parameters specific to this instance of piece from the JSON
        public void getRules()
        public Map<String, Double> getStats()
                Map of stat names as keys and stat as value
        public List<Pieces> getItems()
                Gets list of the pieces this piece contains (ie. Treasure Chest contains treasures)
        public void move()
        public void stop()

//Represents a Patch
public class Patch
        String patchName, int patchID,
        public Patch()
        public Patch(Patch original)
        //clone a patch that has previously been created
//Represents the Grid
public class Grid
        public Grid(int rows, int cols)
                // takes in user-set dimensions and populates grid with null patches
```

```
        public void setPatch(int row, int col)
                // allows user to replace a patch at any row or col with a new patch
                // throws exception if out of bounds row or col is entered

//Contains and checks the rules of the game
public interface Rules
        public boolean isTurnEnd()
        public boolean isLevelEnd()
```

Game Data

The JSON Writing and Parsing are in the Authoring Environment and Game Engine sections.