Team: MISSINGNO

OOGASALAD - Initial Design Document

I. GENRE

Our platform will support the creation of turn-based strategy games. Such games involve players placing units on a map and strategically controlling their behavior. A victory condition such as controlling a given number of points or eliminating enemy units determines when the game is completed. Features of this genre that our API will support include:

- Definition of victory conditions. The designer will be able to create and apply a variety of conditions that define how the game is completed and who wins.
- Creation of a variety of mobile units. Each unit will have specific behavior and attributes.
- Creation of custom map tiles and map layout design. The designer can define the exact way in which a map is organized to support game customization. Further, different tiles may interact with units differently.
- Definition of "turns": what can happen during and what ends a player's turn.

II. DESIGN GOALS

- Allow developer to assign sprite images to objects.
- Allow developer to build map (terrain tiles, size, etc.)
- Allow developer to define objects with properties from a set list (object mines resources, object speed, etc.)
- Allow developer to define objective to win game (collect x resources, conquer x flagpoints, create x units etc.) in order to guarantee flexible framework
- Developer can define clearly how the game is played (what strategies one employs) due to his/her ability to define the map as an author.
- Program a robust AI, able to react intelligently to different developer-defined goals.

Assumptions

- Minimum map size (1-tile)
- At least one player
- At least one mobile unit

Structural Components

The key structural components of a turn-based game are: the map layout, the functions of various units and how they are created, and the objectives for that game.

Game Variety

Our platform will allow for great variety in games designed mainly through these game elements:

Map layout - the configuration of different tiles (impassable, farmable, etc.) creates different points of contest for players. Perhaps resources are central on the map, prompting early aggression, or they are on the fringes, encouraging a late-game.

Game objectives - designers will have free reign over what dictates the end of the game: king of the hill, kill all enemies, survival, etc.

Unit abilities - allowing the designer to set the attributes (health, base attack, etc.) and abilities of all units and the steps to create them. Different unit configurations make for drastically different gameplay.

III. PRIMARY CLASSES AND METHODS

Game Engine

Player Class: contains resources and collection of units.

Map Class: contains the tiles that constitute the current grid the player is working on.

Tile Class: contains information about a given tile coordinate on the map.
giveResources - increment player resource count and decrement tile's resource count.

Unit Class: attributes of any unit that may be placed on a tile (mobile unit, spawner)
[Has a Condition Object associated with it, which specifies what must have been done to create this unit]
move - change tile location on the map.
attack - deal/receive damage from an opposing (or allied?) unit.
harvest - collect resources.
build - create other units.
modify - alter attribute of a unit.
interact - perform action with another unit.
interactedWith - perform action with another unit.
useAbility - create and activate an ability class (likely via Reflection).

Ability Class: performs a wide array of designer-defined actions.
performAction - abstract method carried out by unit's useAbility.
NOTE: not fully sure how we'll implement abilities at this point (i.e. how does user dynamically create a unique Ability class… with a combination of pre-set allowable modifications?).

Condition Class: a set of modular definitions that allow for a certain action (e.g. build a unit, finish the game):
waypoint - move a unit to a particular coordinate.

defeat - kill or otherwise remove a particular unit or set of units
create - must create a given unit or set of units.
resources - must have a certain number of resources.

AI Class: interacts with the given state of the map and makes an intelligent move in opposition to the player. Maintains a priority queue of possible actions.

rankActions - assign values to a given sequence of actions and adds to collection.
performAction - implements the highest-calculated priority in it's collection.

Game Authoring Environment (GAE)

AuthorInterface - an interface implemented by largest Swing container to define API for interactions with other teams.
Various Swing Components to create the GUI. (See mockups in Diagrams section).

Game Data

Encoder Class: gets data from the Game Authoring Environment and converts it into an XML file.
Parser Class: when called by the game player, parses the XML file to provide the necessary information to the player.

XML file example:

```
<xml version="1.0" encoding="UTF-8"?>
<map>
    <tile>
        <passability>1</passability>
        <image>src/images/grass.gif</image>
        <resource>100</resource>
        <type>2</type>
    </tile>
    <tile>
        <passability>0</passability>
        <image>src/images/water.gif</image>
        <resource>20</resource>
        <type>1</type>
    </tile>
</map>
```

Game Player

GameConstructor Class: Creates an instance of JGame (JGEngine) that has the properties described in the file provided by the Game Data team. Will call UnitFactory methods.

Methods - initializeGame(), defineImages(), loadUnits(), setWinningCondition(), setTiles()

UnitFactory Class: Creates a unit to be displayed in the game (and kept track of)
Methods - initializeUnit()

Player Class: Defines how the game is to be interacted with (ex. if P1 is active then only keys A,Z,S,X etc will effect the game, while a different set of keys would be used to interact with the game when P2 is active.)
Methods - defineControlKeys()

GameManager class: Regulates the passing of turns, checks whether winning condition has been accomplished by either (any) player, keeps track of score(s)
Methods - nextPlayerTurn(), checkWinningCondition(), exitGame(), pauseGame()

IV. EXAMPLE CODE

Our final platform will be able to create analogous versions of the following turn-based games:

- Civilization V – this is likely the most in-depth game that will be catered to. A game such as this will have further modification with different game types, such as king of the hill or elimination. The most intensive part of designing this from our platform would be creating tech trees (or the in-game hierarchy of units that may be spawned).
- Chess – the most significant feature of Chess is (obviously) movement of the pieces – whereas most Turn-based games will have units' movement limited by distance and the pass-ability of a given tile, Chess pieces have set, allowed movement schemes.
- Farming Simulator – this is an example of a turn-based game in which there is no opponent. Rather, in this case, the computer AI might randomly introduce conditions impacting the user's play, such as introducing a swarm of locusts. Further, multiple players *could* be incorporated in which the players may cooperate or compete.

V. DESIGN

Our current expected high level flow of information:
The engine acts as a class library from which the GAE draws the information that makes up a unit or tile. This information is passed to the Data team, which formats the information to be used by the Player. The Player then populates a JGame Canvas by creating instances of Unit Objects (whose information was drawn from by GAE).

Alternate Approach:
You will notice that most communication between the groups is one-directional; it is conceivable that other approaches would have groups sending information both ways. For example, while the data group must send the information the player will build the game from, perhaps the player receives incomplete information from data and requests the missing elements (under our current design, we hope to make this scenario impossible).

VI. TEAM ROLES

Game Authoring Environment (GAE)

- Bradley Sykes - set up edit game perspective, map tiles, victory conditions, toolbar functionality, interaction with game data team.
- William Shelburne - set up GUI flow from title screen to edit game perspective, objects, players, interaction with game engine team.

Game Engine

- Timo Santala - map, tiles, AI.
- John Godbey - units, conditions.

Game Data

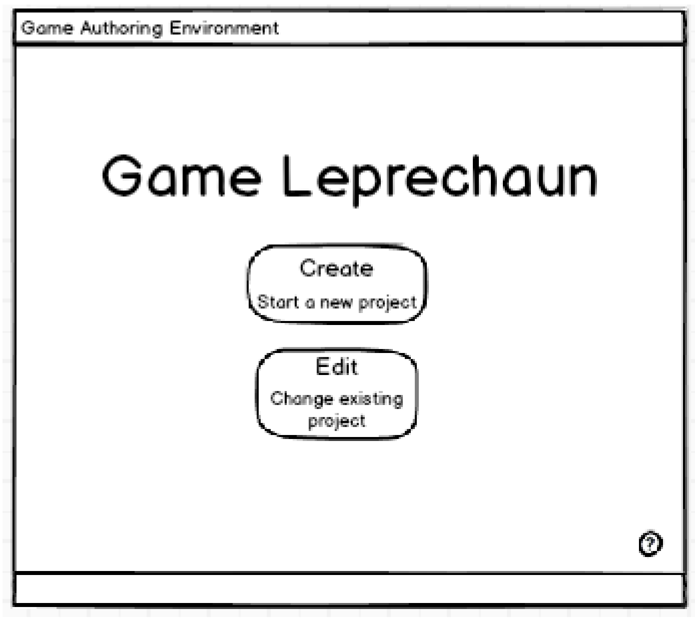
- Alex Song - design and implement the encoder that translates information from the game authoring environment to XML files.
- Seunghyun Lee - design and implement the parser that converts the XML to information that the game player can access.

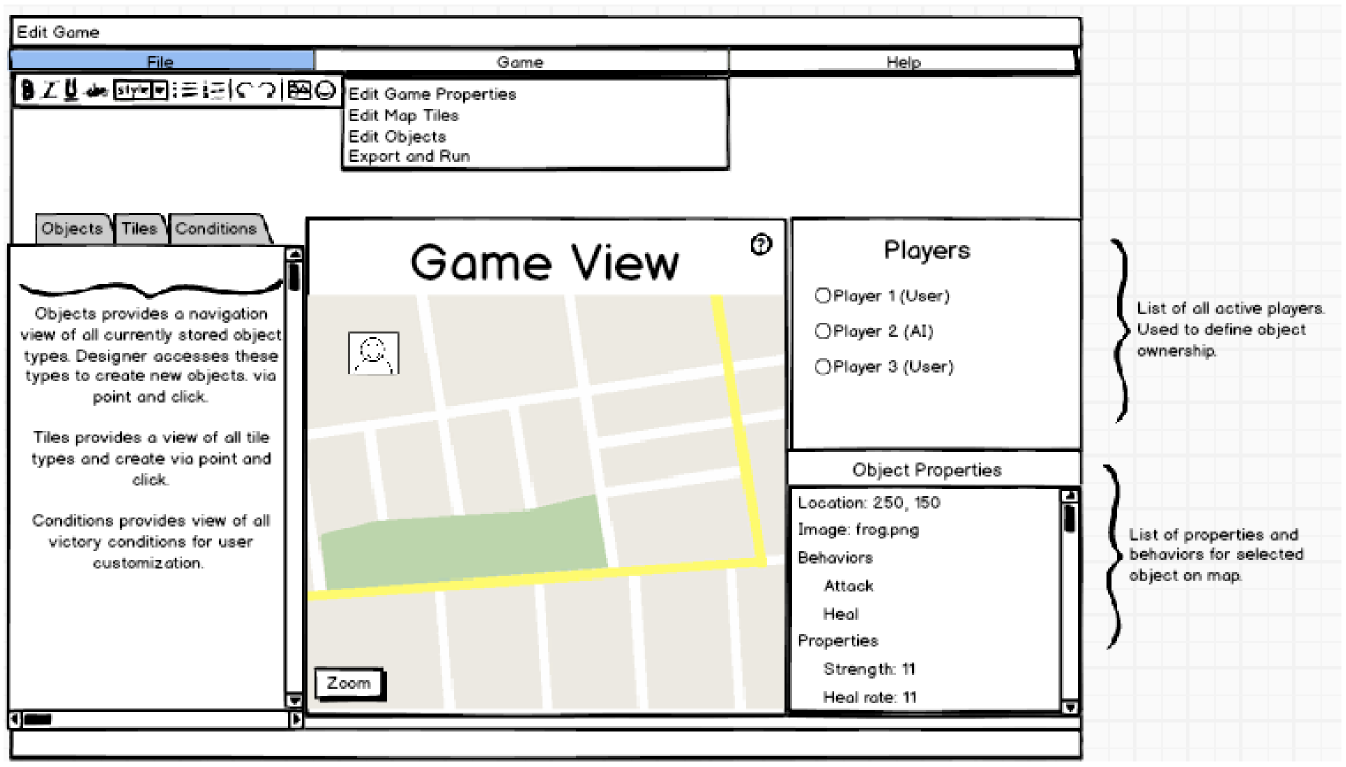
Game Player

- (Both members were not able to attend our Design Document meeting: specific roles will be determined at a later date.)

VII. DIAGRAMS

UI Mockups





Objects provides a navigation view of all currently stored object types. Designer accesses these types to create new objects. via point and click.

Tiles provides a view of all tile types and create via point and click.

Conditions provides view of all victory conditions for user customization.

Location: 250, 150
Image: frog.png
Behaviors
Attack
Heal
Properties
Strength: 11
Heal rate: 11

List of all active players. Used to define object ownership.

List of properties and behaviors for selected object on map.