Team Rocket API Design Plan

Compsci 308 Final Project

Team Members:

Chinmay Patwardhan
Tyler Nisonoff
Ryan Toussaint
Austin Ness
Michael Marion
Matthew Ray
Wes Koorbusch
Robert Ansel

November 16, 2013

OOGASalad Design Document
COMPSCI 308 | Fall 2013

---

http://www.cs.duke.edu/courses/cps108/current/assign/04_oogasalad/part2.php

---

Genre
A 2D RPG in the style of Pokemon. There would be a 2D wandering world mode, where the player could interact with other NPCs, enter buildings/caves/grass, pick up items/etc. There will also be a turn-based battle mode where the player can match up his/her monsters against an enemy's monster or a wild monster.

Monsters will all have dynamic stats related to health, strength, endurance, speed, accuracy, special, and so forth. These stats may improve with good performance in battles. i.e. leveling up of monsters with a certain number of experience points. We may incorporate evolution of monsters at certain levels.

Players may also "catch" wild monsters and add them to their lineup, or add monsters in other ways in the game.

Design goals
The user will be greeted with a GameWizard that walks him/her through the process of a making a game. Each step will be represented by a panel and will deal with creating levels, the main player, monsters, obstacles, NPCs, items, etc. All previous panels will be available at anytime incase the user, for example, wants to create a new monster of extend its functionality. The user will not be able to define modes because only Game and Battle mode will be available. The game will have the ability to be saved and opened in JSON format.

Modules

- **Game Editor:**
    1. **Main Player**
        i. Images (up, down, left, right)
    2. **Type definition**
        i. Define all Types
        ii. Define type matrix, i.e. which types are more or less effective against other types
        iii. Statistics will be hard-coded: attack, defense, accuracy, HP, Experience Points, and Level. However, the user can define status ailments such as Poison which will affect the Monster's statistics every turn of a battle. Default status is OK.
    3. **Attack Wizard**
        i. Define an attack by its name, its power, and any potential statistic

changes (for self or for opponent). e.g.- an attack may increase the user's Attack, decrease the opponent's Accuracy, etc.

4. **Monster Wizard**
    i. Specific definition to build concrete monsters
        - First assign base stats
        - Next assign attacks and at what level they are acquired
            - may be chosen from previously defined move bank
            - may be newly created moves
                - Define effect of move and name
        - Next assign possible evolutions
        - Next assign a catch rate (Number from 0 to 1): affects chance that a monster will be successfully caught in the wild
        - Image
    ii. As monsters are created, all monsters will be visible in a panel
        - These may be edited/deleted
        - These may be linked to define evolutions
            - Must define what level the evolution takes places
5. **Item Wizard**
    i. May be used in battle mode, in wandering mode, or in either mode
    ii. May be used on KO-d monsters, conscious monsters, or both
    iii. May affect a specific monster or all of them
    iv. May recover health, improve stats (temporarily in a battle or permanently), increase a level, heal a status impairment, etc.
6. **Key Item Wizard**
    i. A key item is nothing more than a name. Certain obstacles will require the Player to possess a specific key item (or multiple) to get through. For example, a Rock might block a path and require the user possess all 8 gym badges to get through.
7. **Obstacles**
    i. Image
8. **NPCs (non-fighting)**
    i. Images (up, down, left, right)
    ii. Direction
    iii. Dialogue
9. **NPCs (fighting)**
    i. Images (up, down, left, right)
    ii. Direction
    iii. Dialogue (before fight)
    iv. Post dialogue
    v. Line of sight distance (If the player crosses the NPCs path, the NPC will stop the player and fight them)
    vi. Party of monsters (monsters' names and levels)
    vii. List of key items that the Player will receive for winning (this list may be

empty and probably will usually be empty)

viii. Bet money (amount of money user will gain/lose for winning/losing)

10. Wild Regions

i. Image

ii. Frequency of monster appearing (e.g. once every 15 steps on average)

iii. Monsters and their levels and their probability of being the monster that appears. Probabilities must sum to 1)

11. Level Editor

i. Tabs to define what type of objects to be added. Clicking on a tab will change the right side vertical menu of specific things that may be added

- Obstacle
  - Optional: key items required to pass through
- NPC (non-fighting)
- Fighting NPC
- Wild Region

12. **Fudge factor (0 - 100%)**

i. User defines a global constant that defines the amount of randomness in the game. Recommended value of 5-15%

- For example, an attack will not always do exactly the same amount of damage
- For example, a level 5 Pikachu won't always have exactly the same HP and other stats

- **Game Engine:**

1. **Game Modes (*Two separate views)***

i. World

ii. Battle

- Trainer battle
- Wild battle

2. **Attack Judge**

i. Accepts an attacking monster, victim monster, and Attack. It applies the appropriate effects of the Attack on both monsters

3. **Experience Judge**

i. Player's monsters can ask how much experience they gain based on beating the opposing enemies

ii. Player's monsters can ask how much experience they need to upgrade/level up (size of XP gap)

4. **Abstract Model Objects**

i. **Abstract Visible Game Objects**

- AbstractCharacters
  - Player (the main character)
  - NPC (default non-fighting)
    - Extended by FightingNPC

        - Obstacles
        - Wild Regions
    ii. Monster
    iii. Attack
    iv. Item
    v. KeyItem

(MICHAEL) REVISED TO REFLECT WIZARD API CHANGES

Primary classes and methods for each module

- ***Wizard***
    - *A JDialog with a CardLayout attached, allowing the user to switch between different WizardPanels.*

- ***Wizard Panels***
    - *In the authoring environment, each panel represents a stage in the process of creating a specific element within the game. Each **WizardPanel**, which will extend this abstract class, will be **capable of generating JSON.***
    - *These panels will be dynamically generated and added to a new Wizard each time using reflection to instantiate classes matching to strings in the properties file.*
    - *The **FinishPanel** is the last panel and will be hard-coded into each wizard. It will collect the data from the previous panels in the wizard and write it out to a JSON object.*

  How a WizardPanel generates JSON
    - Each WizardPanel contains some form to fill out specific information for that given state
    - In a properties file, we will outline what a given wizard will have to generate JSON for
    - Each wizard will also map these strings to fields in the form
    - In order to generate the JSON, we will loop over each string in the properties file for a given WizardPanel, use reflection to instantiate an instance of that generic panel, and launch the finished wizard.
    - We will then look at the data inputted into each field that maps to the matching string in the properties file, and generate an JSON tag for it

- **AuthorView**
    - *A superclass housing the entire authoring engine view.*

BACKEND PACKAGES AND CLASSES

Game.Model

- AbstractCharacter - *a class extending AbstractViewableObject used to define character movement - Player extends this class*
- AbstractItem - *an abstract class used for the Item hierarchy*
- AbstractModelObject - *a class defining behavior for all objects within the model. The object at the top of the inheritance hierarchy for objects in the model.*
- AbstractViewableObject - *a class extending AbstractModelObject that defines behavior for all viewable objects in the model.*
- FightingNPC - *the FightingNPC extends the normal NPC behavior*
- GameModel
- Item
- KeyItem - *The **KeyItem** class is used to create*
- Monster - *The **Monster** class is the object which all fighting characters carry in their party for use in Battle Mode. The main player and fighting NPCs will have up to 6 unique Monsters, each with their own attacks and HP, attack, and defense ratings. Monsters can also be created that can evolve at certain levels into different Monsters. Monsters are instantiated by their Player and FightingNPCs as well as WildRegions, and will also communicate with the AttackJudge and ExperienceJudge during and after battle.*
- NPC - *The **NPC** class is used to create non-player characters that are located throughout the wandering mode of the game. The **Player** is able to interact with the NPCs which may include:*
    - *Battling*
    - *Accepting KeyItems*
    - *Learning game knowledge through dialogue*
- Obstacle - *The **Obstacle** class defines behavior for an Obstacle object that can be placed in the game. The Obstacle can also have interaction behavior with the Player and their KeyItems in which the Obstacle can remove itself visually from the game.*
- Player **-** *The **Player** class is the main object that the game revolves around. It contains a collection of monsters that are held in the player's party. The **Game Modes** will only changed based on a specific interaction from the player's viewpoint (e.g. engaging in a battle with another monster). The **Game View** will be communicating heavily with the **Player** via the **Model** and **Controllers** to update its position and know what is visible to the **Player**.*
- Stat
- WildRegion

- World

Game.Model.Attack

- AbstractAttack

Game.Model.Judge

- AbstractJudge
- AttackJudge
- ExperienceJudge

Game.View

- GameView
- Painter

JSONCache

- JSONCache - *This class creates an giant JSON object that stores severals lists of other JSON objects. There may be several different types of JSON objects that it stores, so it keeps track of various categories. Therefore, this giant object basically serves as a cache for different categories of other objects. For example, categories may consist of Weapons, Players, and Items. Each of these lists is a JSON Array. This giant JSON object also has a toString() method to represent itself as a JSON string, so it can easily be written to a file or inserted within one.*
- JSONException - *Exceptions for JSON-related errors*

Location

- Direction - *Enumerable describing direction of Up, Down, Left, or Right*
- Loc - *This is a simple class to represent an x,y coordinate. It also knows how to get an adjacent location in any Direction*

Reflection

- MethodAction
- Reflection
- ReflectionException

    -

- **Model**
    - *Our general **Model** class to handle shared information between all objects on the backend.*

- **“Controller” Package**
    - *Classes in this package define the modes of communication between the model and the view. Its main purpose is to maintain organization so that classes are not compromised by unnecessary communication methods.*
    - *For example, our World View only needs to know about the objects viewable on*

*the current screen. Thus, in our **WorldController**, we have a **getViewableObjects()** which will return a collection of **AbstractViewableObjects**. We believe this is good design because the model should not be concerned with which objects are within some viewable region.*
    - *We also have a **BattleController** class to handle the model-view interactions during the Battle Mode*

- **Monster**
    - *Non-NPC creatures that the player may interact with, whether by adding them to their party or battling them.*
    - *Has a collection of **Attacks** in addition to two required properties: a **health level** — which defines the current number of hit points remaining on the creature — and **catch rate** — a multiplier that helps determine the likelihood of a creature being added to the player's party.*
    - *Based on the user's creation of the game, Monsters may have additional **stats** including, but not limited to, Speed, Attack, Defense, Magic, Special, and so forth.*
    - *Finally, **Monsters** may **evolve** throughout the game based on a system of powering up or gaining experience through battle, which is controlled by the **ExperienceJudge** class.*

JSON Code Examples:
Example Game: Pokemon

Here we outlined the Pokemon Bulbasaur, establishing its attacks (e.g. cut, vine whip, and razor leaf), the level that each is unlocked at, and Bulbasaur's HP and Attack statistics. For each attack, it describes who the attack is directed at (itself or the other monster), that statistic that the attack affects, and the power it has. Actual attack damage will depend on the attack's power, the attacking monster's Attack statistic, the targeted monster's Defense statistic, etc.

```
<Attacks>
      <Attack>
            <Name>Cut</Name>
            <Effects>
                  <StatisticEffect>
                        <Target>other</Target>
                        <Statistic>HP</Statistic>
                        <Power>6</Power>
                  </StatisticEffect>
            </Effects>

            <Name>Vine Whip</Name>
            <Effects>
```

```
                <StatisticEffect>
                    <Target>other</Target>
                    <Statistic>HP</Statistic>
                    <Power>15</Power>
                </StatisticEffect>
            </Effects>

            <Name>Razor Lead</Name>
            <Effects>
                <StatisticEffect>
                    <Target>other</Target>
                    <Statistic>HP</Statistic>
                    <Power>20</Power>
                </StatisticEffect>
            </Effects>
        </Attack>
</Attacks>

<Monsters>
        <Monster>
            <Name>Bulbasaur</Name>
            <FullBaseHP>160</FullBaseHP>
            <CatchRate>.45</CatchRate>
            <Attack>3</Attack>
            ...
            <Attacks>
                <Attack>
                    <Name>Cut</Name>
                    <UnlockLevel>2</UnlockLevel>
                </Attack>

                <Attack>
                    <Name>Vine Whip</Name>
                    <UnlockLevel>6</UnlockLevel>
                </Attack>

                <Attack>
                    <Name>Razor Leaf</Name>
                    <UnlockLevel>15</UnlockLevel>
                </Attack>
                ...
            </Attacks>
            <Evolution>
```

```
                <Monster>Ivysaur<Monster>
                <Level>16</Level>
            </Evolution>
      </Monster>
      ...
</Monsters>
```

Example Game: Duke Basketball Game
Here we outline two attacks, Dunk and Stomp. Dunk is an attack that multiple basketball players will use whereas Stomp is more specific to our Christian Laettner Monsters. Evolutions are correlated to school years, shown by Freshman Christian Laettner versus Senior Christian Laettner.

```
<Attacks>
  <Attack>
    <Name>Dunk</Name>
    <Effects>
      <StatisticEffect>
        <Target>other</Target>
        <Statistic>HP</Statistic>
        <Power>-2</Power>
      </StatisticEffect>
      ...
      <StatusEffect>
        <Target>self</Target>
        <Status>Swag</Status>
        <Duration>5</Duration>
      </StatusEffect>
      ...
    </Effects>
  </Attack>
  <Attack>
    <Name>Stomp</Name>
    <Effects>
      <StatisticEffect>
        <Target>other</Target>
        <Statistic>Defense</Statistic>
        <Power>-1</Power>
      </StatisticEffect>
      ...
      <StatusEffect>
        <Target>other</Target>
```

```
        <Status>paralyzed</Status>
        <Duration>1</Duration>
      </StatusEffect>
      ...
    </Effects>
  </Attack>
  ...
</Attacks>

<Monsters>
  <Monster>
    <Name>Freshman Christian Laettner</Name>
    <FullBaseHP>70</FullBaseHP>
    <CatchRate>.01</CatchRate>
    <Attacks>
      <Attack>
        <Name>Dunk</Name>
        <UnlockLevel>32</UnlockLevel>
      </Attack>
...
    </Attacks>
    <Evolution>
      <Monster>Sophomore Christian Laettner<Monster>
      <Level>32</Level>
    </Evolution>
  </Monster>
  .<Monster>
    <Name>Senior Christian Laettner</Name>
    <FullBaseHP>0</FullBaseHP>
    <CatchRate>.01</CatchRate>
    <Attacks>
      <Attack>
        <Name>Stomp</Name>
        <UnlockLevel>70</UnlockLevel>
      </Attack>
...
    </Attacks>
  </Monster>
..
</Monsters>
```

Design alternatives

Representation of data was an issue — we debated whether or not to use XML or JSON; we

decided on JSON because we found it easier to parse and write. We also had to decide on the elements in the RPG genre that were specifically related to Pokemon and whether or not we wanted to implement them. We tried to generalize the idea of “capturing” other characters and adding them to a “party” instead of simply having “Poke Balls” capture other creatures.

Using the MVC model, we debated on whether or not to include direct Controller classes; in the end, we decided to include these classes because they isolate the logic used to communicate between the front-end and back-end systems, preserving the integrity of the design while maintaining flexibility and scalability.

Roles and responsibilities
Chinmay Patwardhan - backend and frontend
Tyler Nisonoff - backend
Ryan Toussaint - backend
Austin Ness - backend
Michael Marion - frontend
Matthew Ray - frontend
Wes Koorbusch - frontend
Robert Ansel - frontend

Assumptions:

- One controllable player.
- Battles will take place between two team leaders and their corresponding parties.
- Battles have a variable winning condition — not just hit points.
- **The user should not have to type code when using the authoring environment**

UML Diagram
The diagram below represents our current class hierarchy. It outlines the major methods within each class and how they are related (e.g. which classes extend which). Please see our online repository for a more detailed version of the UML diagram (API_Class_Diagram.gif).

Team Name: Team Rocket

Members:
Chinmay Patwardhan
Tyler Nisonoff
Ryan Toussaint
Austin Ness
Michael Marion
Matthew Ray

Wes Koorbusch
Robert Ansel

Idea: Pokemon type of RPG. There would be a 2D wandering world mode, where the player could interact with other NPCs, enter buildings/caves/grass, pick up items, etc. There will also be a turn-based battle mode where the player can match up his/her monsters against an enemy's monster or a wild monster. Monsters will all have dynamic stats related to health/strength/endurance/speed/accuracy/special/etc. These stats may improve with good performance in battles. i.e. leveling up of monsters with a certain number of experience points. We may incorporate evolution of monsters at certain levels. Players may also "catch" wild monsters and add them to their lineup, or add monsters in other ways in the game.

Goals:
- Example: Beat all the gyms (in chronological order?)
- Example: One or more NPCs who represent the "Elite 4." Goal is to beat these NPCs in a battle
- We can abstract this to generic "Challenges" that must be beaten to advance to new stages in the game.

What will be available in the authoring environment?
- Draw entire world with obstacles/buildings/grass/water/etc
- Define challenges and how they are completed. When they are completed the Player data should reflect that so that new portals will open or obstacles will disappear.
- Define portals which may restrict access if challenges haven't been completed
- Define areas where wild enemies may appear and their frequency
- Add monsters with base stats with a fudge factor. Define attacks.
- Add NPCs with or without monsters. Define dialogue.

Contact Information:

Ryan Toussaint	(248)-882-1795	rmt15@duke.edu
Chinmay Patwardhan	(703) 485-5298	cmp49 / chinnychin19@gmail.com
Michael Marion	(919) 749 - 8846	mtm36@duke.edu
Austin Ness	(703) 862 9828	aen8 / austinness@gmail
Tyler Nisonoff	(201)-484-9014	tcn2 / tylernisonoff@gmail.com
Wes Koorbusch	(203) 940-1940	twk8 / wes.koorbusch@gmail.com
Matthew Ray	(404)-754-6024	mjr19@duke.edu