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 1, 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 XML format.

Modules

- **Game Editor:**
    1. **Type definition**
        i. Define all Types
        ii. Define type matrix, i.e. which types are more or less effective against other types
    2. **Monster definer**
        i. Generic definition first (applies to all monsters): what stats are available
        ii. Specific definition to build concrete monsters
            - First assign base stats
            - Next assign moves
                - may be chosen from previously defined move bank
                - may be newly created moves
                    - Define effect of move and name

iii. 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

3. **Item definer**
    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
4. **Level creator**
    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
            - Rock, Flowers, Vertcial Fence, Horizontal Fence, Tree
            - Previously created Portals
            - ‘New’: define name, image, whether or not it’s a Portal
                - If it’s a Portal, define what checkpoints need to be present to pass
        - Building
            - Hospital
            - List of previous markets
            - ‘New Market’: pops up definition for what items to be sold
            - ‘New building’: pops up new level creator window to define inside of building
        - NPC
            - List of names of previously created NPCs
            - ‘New’: define name, monster list, dialogue message
        - Wild Region
            - List of names of previously defined regions
            - ‘New’ region: choose image, name, frequency of monster appearing, specific probabilities of which monsters (and their levels) will appear. Probabilities must sum to 100%
                - When choosing monsters, an option for ‘New’ to add new monsters will be available
        - Item
            - List of items
            - ‘New’
5. **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. **Parser**

    2. **Game Modes (*Two separate views)***
        i. World
        ii. Battle
            - Trainer battle
            - Wild battle

    3. **Inventory (Of game items)**
        i. Items: potions/powerups/etc
            - Items have a value (e.g. different for buying in a mart vs. player selling it)
        ii. Key Items: "Access Card"/Badges/"Chainsaw" to cut down trees
        iii. "Poke Balls": Capturing devices to catch monsters
    4. **Wild Terrain**
        i. Defines grass, forests, caves, etc.
        ii. Defines the chance of wild pokemon appearing and their level
        iii. Has a map of which pokemon appear and how often
            - must add up to 100%
    5. **Player**
        i. Main character
            - All players have a party (may be empty though…)
            - Has inventory
            - Interact with game objects
            - Money (e.g. used in battles and at marts)
        ii. NPC
            - Bet Money (e.g. bet made in a battle)
    6. **Monster Types (optional)**
        i. If not defined, default type of Normal will be assigned to everything
        ii. Fire/Water/Rock/etc
        iii. Table defining multipliers between monsters of different types
            - i.e. Fire super effective against grass
            - i.e. Ground does not affect ghost
            - i.e. Grass not very effective against fire
    7. **Attacks**
        i. Health changes
            - This will be passed to a an Object that determines the amount of damage done to a monster. The inputs will be the attacking monster, the victim monster, and the attack used. It will depend on

monster accuracy, monster attack, monster defense, monster type, attack accuracy, etc.

ii. Stat changes
- i.e. effect on accuracy/speed/attack/defense, but NOT health

iii. Status changes (e.g. poison or frozen)

iv. Accuracy (e.g. chance of attack being successful)

8. **Attack Judge**
   i. Accepts an attacking monster, victim monster, and Move. It applies the appropriate effects of the Attack on both monsters
9. **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)
10. **Monster**
   i. Has a status that can be affected by impairments (e.g. poisoned, frozen)
   ii. List of attacks
      - Attack knows how to update the monster's and the opposing monster's stats
      - Attacks have accuracy
      - Attacks of the same Type as the monster may be multiplied by the Special stats
   iii. Stats:
      - Health level (Required) (e.g. determines a winning condition in a battle)
      - Catch Rate (Required)
        - Probability of a catch = CatchRate(ball) * CatchRate(monster) / HealthFraction(monster)
      - Speed, Attack, Defense, Magic, Special, etc. (Optional)
      - ~~Wild vs. not wild (Required)~~ *Handled by the game mode.*
      - Experience (calculated using the Experience Judge)
        - expOnCurLevel
        - expToNextLevel
        - curLevel
        - evolveLevel
11. **Visible Game Objects**
   i. Interactive
      - Characters (Fighters/NPC)
        - Dialogue -- ("Do you want to fight me?")
      - Portals
        - Dialogue -- ("You're entering the Hospital.")
      - Game Items (may be picked up off the ground)
        - Dialogue -- ("You picked up a poke ball.")

        - Map - adds explored areas
    ii. Multi-tiled
        - Caves, Buildings, etc.

12. Hospitals

i. Will all look the same, and be a place where all monsters may be revitalized

13. Marts

i. Will all look the same, but store different lists of items that may be sold

Primary classes and methods for each module

- **AbstractWizardState**
    - *In the authoring environment, each state represents a stage in the process of creating a specific element within the game. Each **WizardState**, which will extend this abstract class, will be **capable of writing XML**.*

- **AuthorView**
    - *Holds a **List** of all possible **WizardStates** and has a public interface **nextState()** and **prevState()** to switch between **WizardStates**.*

- **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**.*

- **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.*

How a WizardState generates XML

- Each wizard contains a 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 xml for
- Each wizard will also map these strings to fields in the form
- In order to generate the XML, we will loop over each string in the properties file for a given wizard, look at the data inputted into the field that maps to that string, and generate an xml tag for it

XML 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 JSON or XML; we decided on XML because more people on the back-end team had experience with the XML file format. 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
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