NVriezen

Revision: 1.0.0

GDD Template Written by: Benjamin “HeadClot” Stanley

Special thanks to Alec Markarian
Otherwise this would not have happened

Overview

Theme / Setting / Genre

- RPG

Core Gameplay Mechanics Brief

- Walking
- Talking
- Collecting
- Battling
- Connecting with Friends

Targeted platforms

- Android

Project Scope

- Game Time Scale
    - 8 weeks
- Team Size
    - Niels Vriezen
        - Programming
        - Game Design
- Software / Hardware
    - Engine
        - Unity 3D Personal
    - Software
        - Audacity
        - Word 2013
        - GIMP
    - Hardware
        - Two Android Devices
        - Laptop capable of running above software
- Total Costs
    - No Costs

Influences

- Pokémon Gold & Silver – Nintendo Game Boy

- This game will be a remake of Pokemon Gold & Silver. Original sprites will be used from the games.

- Hitmonchan Boxing – Pokémon Mini

- A game for the smallest Nintendo handheld, the Pokémon Mini. This game asks the player to shake the device and react on vibrations to attack and defend in a battle. This inspired me for the twist for the retro game.

- Wifi Direct

- This retro game twist assignment give me inspiration to test new technologies. I saw potential in Wifi Direct to be used as a new way to connect devices, instead of using Bluetooth.

The elevator Pitch

Pokemon Shake is a remake of Pokemon Gold & Silver with new technologies applied. The input system has been newly designed for mobile devices. The battle system is replaced, just like the catching mechanic of the original games. The player needs to shake the device and react on the vibrations. Pokémon can only be obtained by connecting with friends.

What sets this project apart?

- New Mobile Specific Input System
- Pokémon can only be obtained by connecting with friends
- First time main series game on mobile device
- New unique battle system

Plan of Action

Engine Specific Features

- 3D Environment
- Sprites together with UI Sprites
- Orthographic Cameras
- Android Platform Module
- Specific Android Functions (vibrating, Wifi Direct)
- Nodes and Grid

How does it work?

- Mostly I will be using the original sprites from the game. These will be implemented via Unity’s 3D sprite system in combination with Unity’s UI sprite system. For the movement I will be using touch input which sets a target position for the player. With A* pathfinding the player object will move to specific Nodes which are located on a grid to bring back the original 2D tile based movement from the original games. Whereas the orthographic camera will be able to display the sprites and other assets in a pixel perfect way. While implementing some Android specific features to be used for the battle system and connecting-with-friends feature.

Priority

- **Short Answer**
    - Managing huge amount of data
    - Sharing data with other devices
    - Battle system with unique features
- **Long Answer**
    - The priority for this project is managing the huge amount of data for the specific Pokémon in the game. Where the player can easily get their Pokémon's information via the in-game menu. Together with being able to connect to other devices and sharing bits of their information with other users. A third priority would be the battle system which is the main twist of the game. Being able to let the device vibrate in a certain pattern and making this interactive with player input.

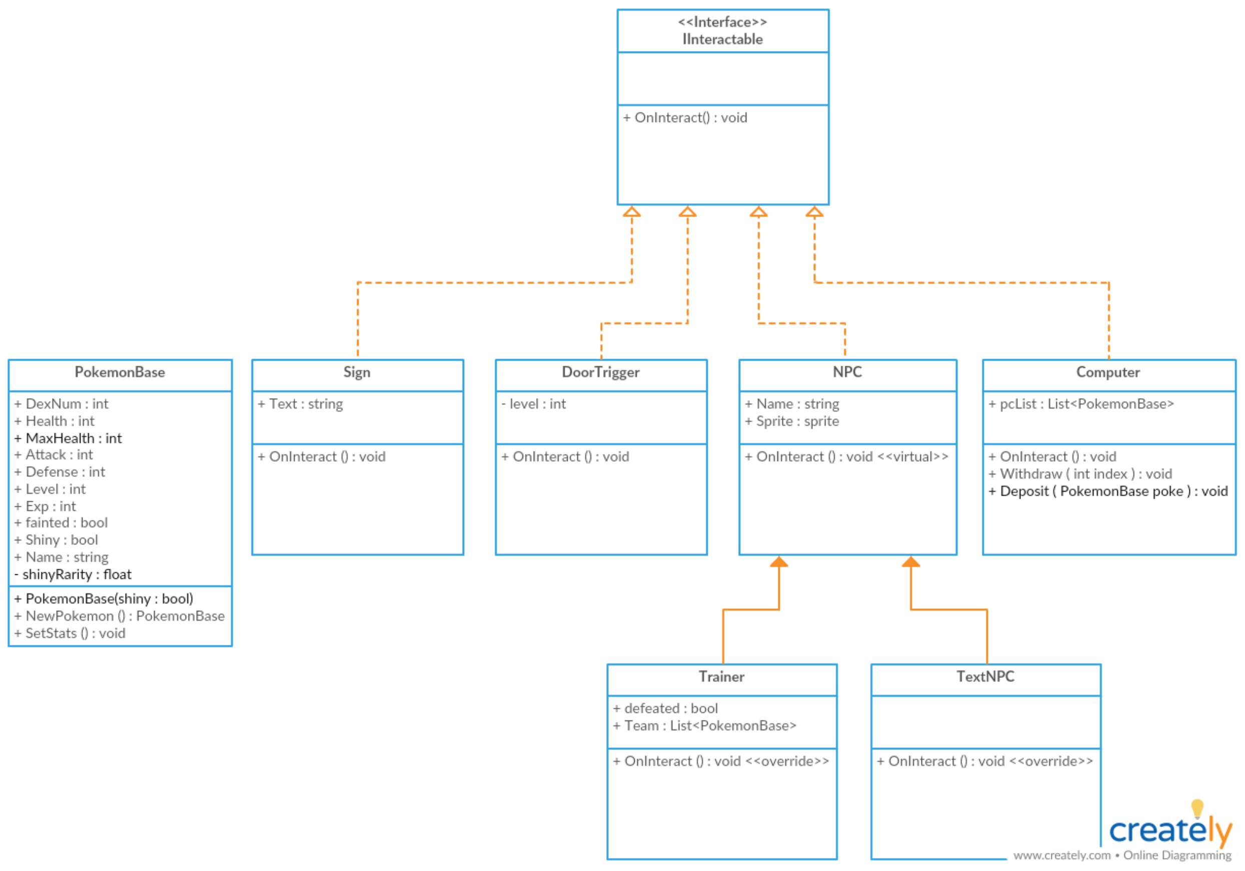
Code Structure

- For the data of the Pokémon I have decided to go with a base class for the Pokémon themselves. This base class can be attached to objects. The class will communicate with a class for JSON file reading. The JSON file will contain all data for the Pokémon. This includes names, numbers, moves, etc. Once a Pokémon gets instantiated, it will gather information which corresponds to the Pokémon class constructors' parameters.
- For the teams I will have a class for the trainer team and a class for the enemy team. The trainer team class will keep all the information of the player's team. It contains a list of Pokémon classes.
- I am planning on using an event system which handles all events in the game. But for now this does not seem to be necessary, but could be once multiple things need to happen when loading a scene. The place where it is needed the most is for the PC system. But since that feature is not prioritized, I will not implement it yet.
- Singletons will be used for Sceneloader and Player object. The Player object holds the data for all the obtained Pokémon. Therefore it should only be made when the game starts. After that it should not be deleted.
- Facade is a design pattern which could work well for the UI system of the game. I am using this currently in a very basic way but want to extend it and implement it in a better way.

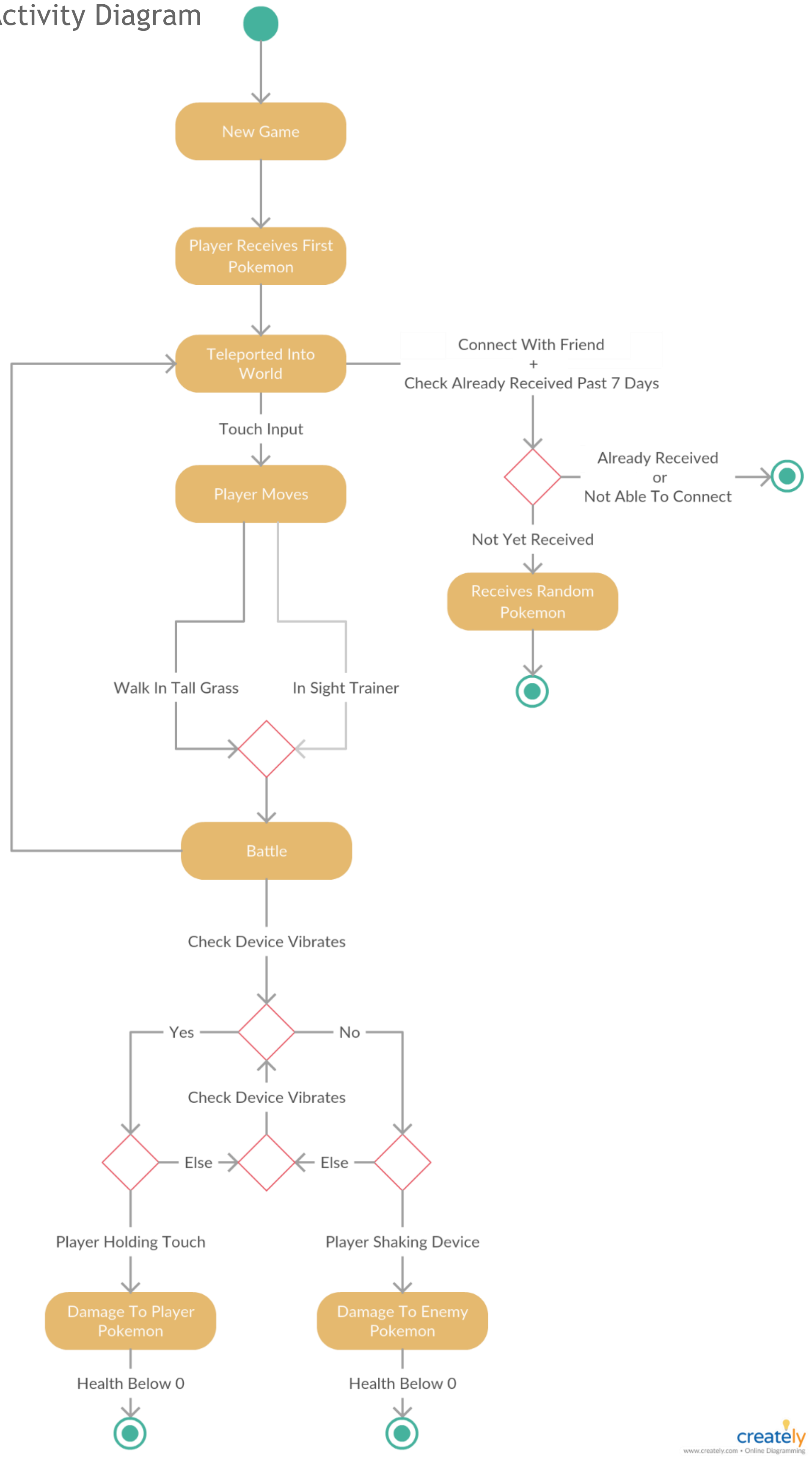
- Decorator pattern will be implemented for NPCs. This way we can extend an NPC with trainer capabilities, without making different classes for it.

UML

- Class Diagram



- Activity Diagram



Code Design
- This code design is for me the current best and doable way to implement all the wanted features. It contains new patterns I have never used before together with some familiar patterns. The familiar patterns used need some more practice and therefore I try to use those as well.
I always try to think about performance. Using such patterns and trying to do things differently than normal, hopefully makes me able to implement easier to use but also more performance friendly code snippets into projects. Building a vocabulary for myself is very important as I do not have enough programming skills and knowledge yet. Using as many patterns in the right way hopefully helps me build this vocabulary.
- My goal is to make the classes easy to use and manageable for myself. Patterns can be a good way to keep communication between classes simple. Together with making it easier to expand behavior of objects with modular classes which use certain design patterns (like Facade or Decorator).