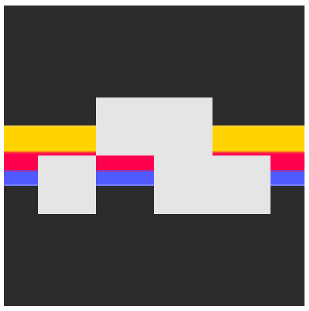


NVriezen

Revision: 1.1.0

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
- 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
        - Word 2013
        - GIMP
        - MonoDevelop
    - Hardware
        - Two Android Devices
        - Laptop capable of running above software
- Total Costs
    - No Costs

Influences

- **- Pokémon Gold & Silver – Nintendo Game Boy**
  - This game is a remake of Pokémon 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

Pokémon Shake is a remake of Pokémon 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

Technical Description

Engine Specific Features

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

How does it work?

- The original sprites from the game have been used to produce this game. These sprites were implemented via Unity’s 3D sprite system in combination with Unity’s UI sprite system. A* pathfinding has been utilized to create a convincing touch input system for movement which complements the old button input system. The target position gets set on the location the user touches the screen. The nearest Node gets set as the destination for the character. Avoiding obstacles through the use of colliders.

Whereas the orthographic camera is able to display the sprites and other assets in a pixel perfect way. Only this time the player is able to see a lot more of the world at once. Creating a true mobile experience. While implementing some Android specific features to be used for the battle system and connecting-with-friends feature. Such as vibrating and WiFi Direct.

Priority

- **In short**
    - Managing huge amount of data
    - Sharing data with other devices
    - Battle system with unique features
- **Complete**
    - The priority for this project was 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. At the end this turned out to be the most advanced piece of coding needed for the game. The focus was therefore laid upon making a working prototype which utilizes a good base to extend the battle system in the future.

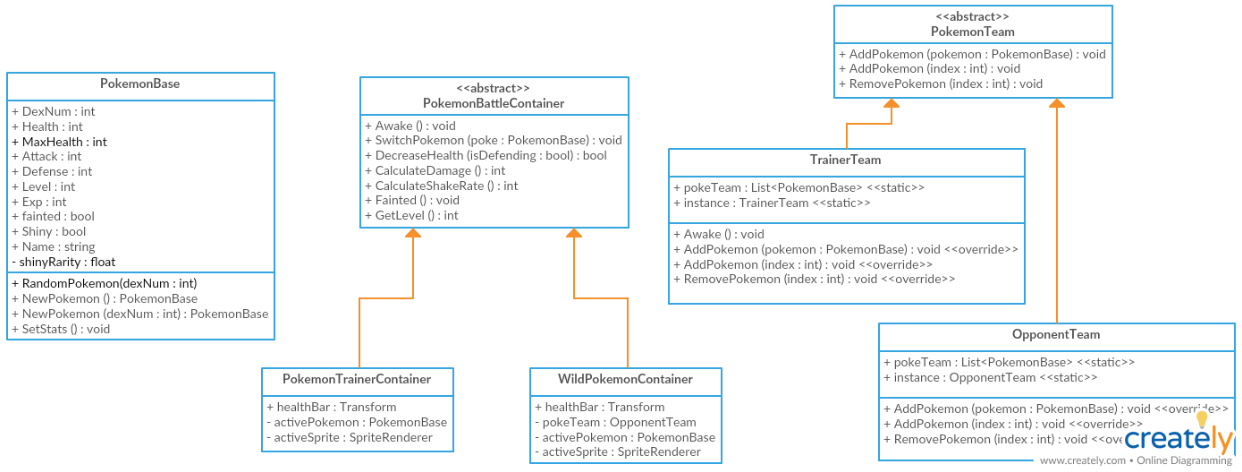
Code Structure

- 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 communicates with a DataManager-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 given dexNumber. A new PokemonBase gets instantiated and all fields will get assigned with the correct values received from the JSON file.
- For the teams I will have an abstract base class which defines the necessary methods. The trainer team and the enemy team classes inherit from this base class and therefore need to implement those methods from the base class. The trainer team class will keep all the information of the player's team together with managing adding and removing Pokémon from the team. This is done through a List<> of PokemonBase components. The same is done in the enemy team class.

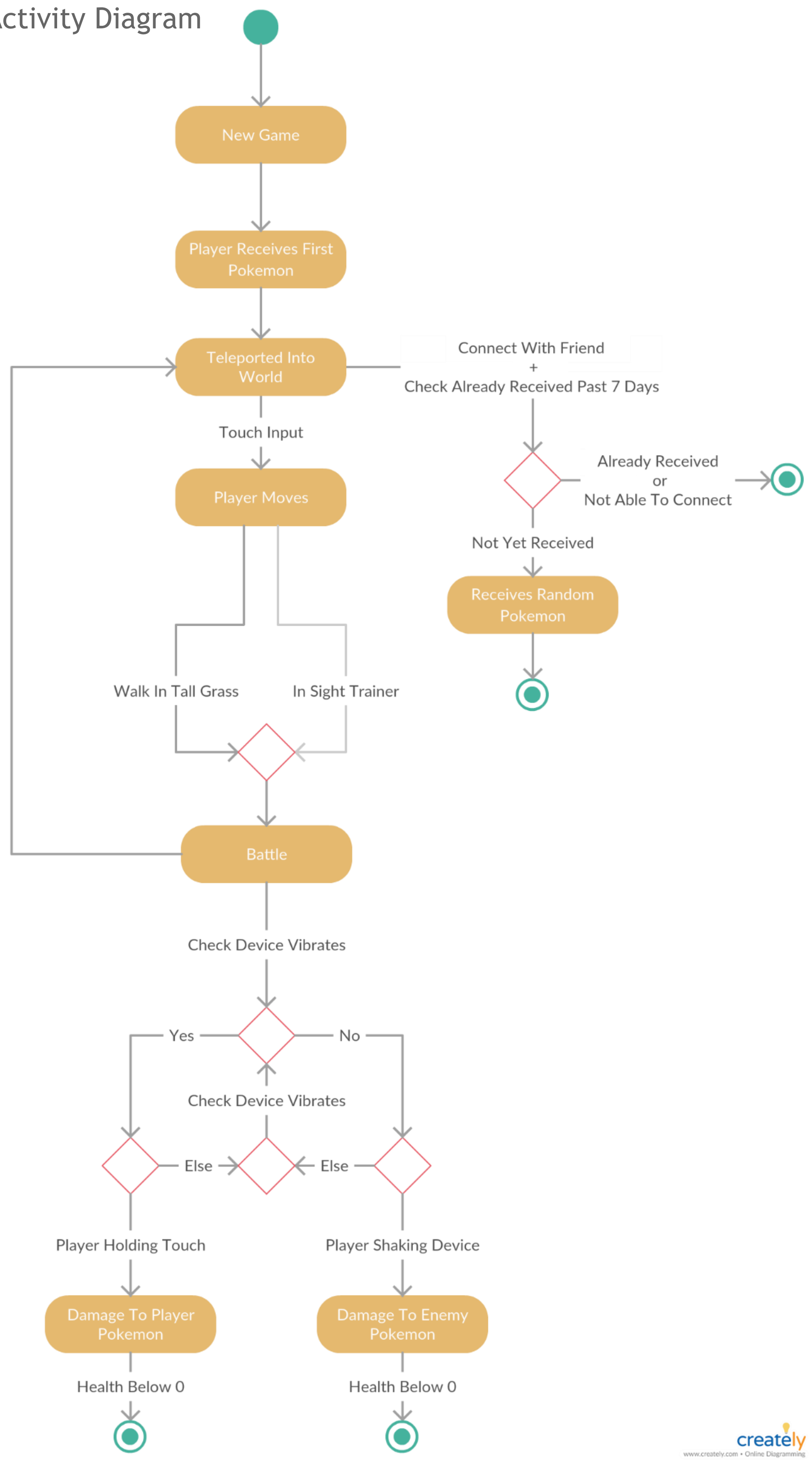
- An event manager has been programmed. Though it has currently not been used. The same for the SceneManager. These were programmed for scene loading. This is currently still done via buttons and triggers.
- Singletons are used in multiple classes. The before mentioned EventManager and SceneManager utilizes this pattern. It could be implemented for the MainCamera as well, but this requires changes in managing certain components attached to the camera.

UML

- Class Diagram


PokemonBase
+ DexNum : int
+ Health : int
+ MaxHealth : int
+ Attack : int
+ Defense : int
+ Level : int
+ Exp : int
+ fainted : bool
+ Shiny : bool
+ Name : string
- shinyRarity : float
+ RandomPokemon(dexNum : int)
+ NewPokemon () : PokemonBase
+ NewPokemon (dexNum : int) : PokemonBase
+ SetStats () : void
<<abstract>>
PokemonBattleContainer
+ Awake () : void
+ SwitchPokemon (poke : PokemonBase) : void
+ DecreaseHealth (isDefending : bool) : bool
+ CalculateDamage () : int
+ CalculateShakeRate () : int
+ Fainted () : void
+ GetLevel () : int
PokemonTrainerContainer
+ healthBar : Transform
- activePokemon : PokemonBase
- activeSprite : SpriteRenderer
WildPokemonContainer
+ healthBar : Transform
- pokeTeam : OpponentTeam
- activePokemon : PokemonBase
- activeSprite : SpriteRenderer
<<abstract>>
PokemonTeam
+ AddPokemon (pokemon : PokemonBase) : void
+ AddPokemon (index : int) : void
+ RemovePokemon (index : int) : void
TrainerTeam
+ pokeTeam : List<PokemonBase> <<static>>
+ instance : TrainerTeam <<static>>
+ Awake () : void
+ AddPokemon (pokemon : PokemonBase) : void <<override>>
+ AddPokemon (index : int) : void <<override>>
+ RemovePokemon (index : int) : void <<override>>
OpponentTeam
+ pokeTeam : List<PokemonBase> <<static>>
+ instance : OpponentTeam <<static>>
+ AddPokemon (pokemon : PokemonBase) : void <<override>>
+ AddPokemon (index : int) : void <<override>>
+ RemovePokemon (index : int) : void <<ov
creately
www.creately.com • Online Diagramming

- 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.
I always try to think about performance. Using such patterns let me think about how problems can be solved in different ways. 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. Together with using new programming terms I never thought about using before, like abstract and protected for example.
- 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).
Though currently some behavior could be implemented in a better way through different patterns. Together with more performance friendly code.

Sources

- Videos
    - A* Pathfinding Tutorial (https://www.youtube.com/watch?v=-L-WgKMFuhE)
- Websites
    - UI Optimization tips (https://unity3d.com/how-to/unity-ui-optimization-tips)
    - SourceMaking (https://sourcemaking.com/design_patterns/)
    - Pixel Perfect Camera (https://forum.unity.com/threads/the-best-pixel-perfect-method.509323/)
    - Pixel Perfect Unity Game (https://hackernoon.com/making-your-pixel-art-game-look-pixel-perfect-in-unity3d-3534963cad1d)
- Assets
    - Original Sprites (https://www.spriters-resource.com/game_boy_gbc/pokemongoldsilver/)
    - WiFi DirectBase (https://github.com/saltyJeff/UnityWifiDirect)
- Inspiration
    - Pokémon Gold, Silver & Crystal (https://www.nintendo.nl/Games/Game-Boy-Color/Pokemon-Gold-Version-266076.html)
- Sound & Music
    - Original Soundtrack (https://downloads.khinsider.com/game-soundtracks/album/pokemon-gold-silver)