Tommy Hollenberg
CS162 Assign4
Design Document
05/21/2017

Understanding the Problem

The purpose of this program is to create a game inspired by Pokémon Go. The object of the game is to collect all the Pokémon, fully evolve them, and return to Professor Oak.

The game is played on a square grid of "locations." The grid size should be specified as a command-line argument, but must be 3x3 or larger. Each location may contain at most one of four "events:" a Poké-stop (which awards the trainer with 3-10 Pokéballs), a cave (which awards the trainer with a megastone), a Pokémon, or Professor Oak. These events are randomly distributed throughout the grid. The trainer starts the game at the location of Professor Oak, and has two options: move to an adjacent grid location or throw a Pokéball (only accessible when the trainer is in possession of Pokéballs and at a location with a Pokémon). When the trainer is adjacent to a location with an event (other than Professor Oak), a "percept" (message) notifies the trainer, but does not state which direction.

There are two Rock Pokémon (Geodude and Onix) with a 75% catch rate, two Flying Pokémon (Charizard and Rayquaza) with a 50% catch rate, and two Psychic Pokémon (Mewtwo and Espeon) with a 25% catch rate. If the Pokémon is not captured, it moves to another random location on the grid. Each Pokémon has three stages of evolution: Basic, Stage 1, and Stage 2. When first caught, each Pokémon is at the Basic stage. After catching a Pokémon-specific number, it evolves to Stage 1. Evolution from Stage 1 to Stage 2 can only be achieved with megastones.

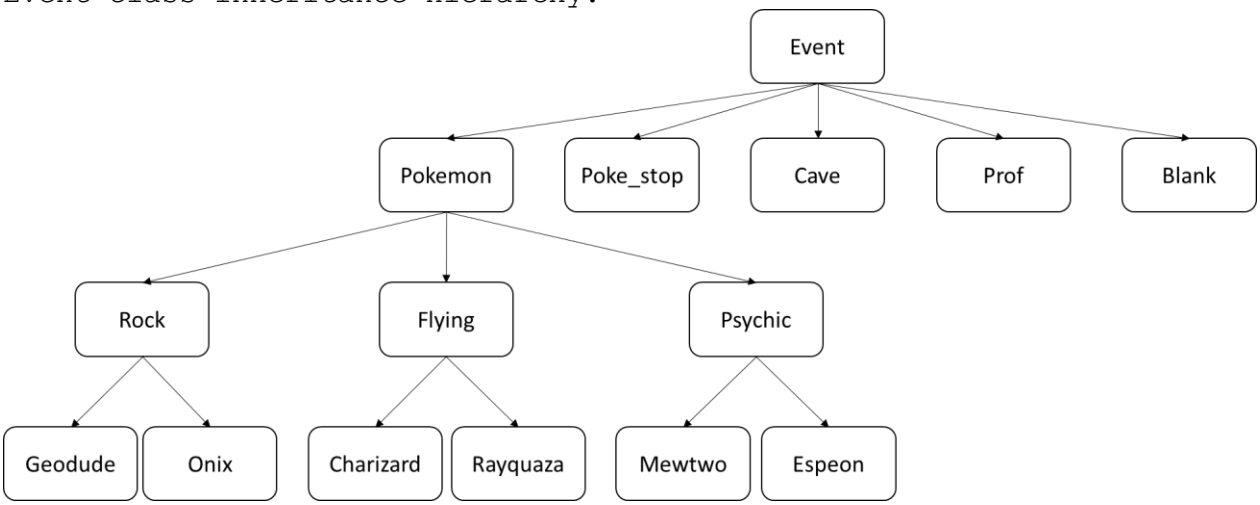
The program should make use of an abstract polymorphic Event class, from which all Pokemon, Poke-stops, and Caves will be derived. Additionally, a Location class that holds one Event must be used. The program should have no memory leaks and all functions should be under 20 lines. Once the user wins the game, they should be given the choice to play again with the same map, play again with a new map (with new dimensions if desired), or quit.

To make this program easier to grade and debug, there must be a debug mode that tells the user what events are in each direction and reduces the caught requirement to one for each Pokemon.

For extra credit, an AI class that plays the game on its own must be created.

Devising a Plan

Event class inheritance hierarchy:



The classes derived from Pokemon will be very simple, and mainly be used to initialize an object with specific values (i.e. the catch rate associated with type, name and number required to evolve to stage 1). The Blank class will contain mostly empty overrides of the virtual Event functions, and will be used as a place holder for Location objects without a more interactive Event.

The abstract Event class will contain three virtual functions:

- Event* clone() – which will be used for polymorphic deep copying,
- void percept() – which will display the appropriate message when the Event is in an adjacent Location, and
- int perform_event(Trainer&) – which will be used to execute the Event-Trainer interactions when the player moves to the Event's Location.

The integer return type of perform_event() will be used to signal that the Event should be moved to a new Location (if it is a Pokemon that was caught or ran away) or that the game is over (if it is Prof and the player met the requirements to end the game).

The Pokemon class is the only Event that contains member variables, some of which are useful for wild Pokemon, and some for the Trainer's Pokemon:

- string name – the name of the Pokemon (relevant for both)
- int stage – the Stage of the Pokemon (only relevant for the Trainer's Pokemon)
- int num_caught – the number caught (only relevant for the Trainer's Pokemon)
- int num_req – the number required to evolve to Stage 1 (only relevant for the Trainer's Pokemon)
- double catch_rate – the chance of success upon throwing a pokeball (only relevant for wild Pokemon)

Additionally, a Game_World class and Trainer class will be utilized.

The Trainer class will contain several member variables:

- int megastones – number accumulated for evolved Stage 1 Pokemon to Stage 2
- int pokeballs – number accumulated for throwing at Pokemon
- Pokemon *pokemon[NUM_POKEMON] – a static array of Pokemon pointers of size NUM_POKEMON (which is a preprocessor macro that, in this iteration of the game, is 6)

When the Trainer catches a Pokemon, the names of the Pokemon already in the pokemon array are compared with the name of the newly caught Pokemon. If a match is found, the num_caught member of that Pokemon is increased (evolution to Stage 1 occurs if the Pokemon was previously Stage 0 and num_caught now equals num_req). Otherwise, a deep copy of the Pokemon is made (with clone()), and its address is assigned to the first open (null-valued) pointer in the pokemon array. The other main functionality of the Trainer class is the ability to view their inventory, which lists the stats of their Pokemon (name, stage, num_caught / num_req, etc.) and their items (megastones and pokeballs), and allows them to use their megastones to evolve any Stage 1 Pokemon. Of course, this assumes that the Trainer has both megastones and Stage 1 Pokemon.

Perhaps the most important class of them all is the Game_World class. The Game_World class contains the following member variables:

- Location **grid – the 2-dimensional array of Locations on which the game will be played
- int length – the number of rows
- int width – the number of columns
- Location *current – the player's current location (starts at Professor Oak)
- Location **blank – an array of pointers to the Locations in the grid that contain Blank events (for wild Pokemon moving purposes)
- int num_blank – the number of Blank events in the grid
- Trainer *trainer – the trainer holding the items and caught Pokemon

The Location class has some additional members that facilitate navigation:

- Event *event – the Event object associated with the Location,
- Location *up – the Location above this Location (if any)
- Location *down – the Location below this Location (if any)
- Location *left – the Location left of this Location (if any)
- Location *right - the Location right of this Location (if any)

In the Game_World constructor, the up, down, left, and right members of Locations in the grid will be assigned to the adjacent Locations. For the Locations on the edges, one (two in the corners) of these will

be null pointers. A vector of string, integer pairs will be populated with Event strings (i.e. “Prof”, “Cave”, “Charizard”, etc.) and their desired abundance on the grid. There will only be one Prof Event, but the number of the other Events varies based on the grid dimensions. The number of each Event will be:

(length * width – 1) / (NUM_POKEMON + 2)

One is subtracted from length * width to account for the Prof, and two is added to the number of Pokemon for the Poke_stop and Cave Events. The remaining Locations will be Blank. The Locations in the grid are looped through in order, and a random number mod the size of the vector is used to determine which Event shall be placed in each Location. The integer value of the selected string, integer pair is decremented, and if it becomes zero, the pair is removed from the vector, decreasing its size by 1 (this is important because the size is used as the modulus for the random number).

During gameplay, a vector of Location*, string pairs is used to hold the move choices for each turn, depending on the location in the grid (for example, some moves are restricted when at the edge of the grid. All four direction (up, down, left, and right) are checked. If they are non-null, the Location’s Event’s percept() function is called, which prints a message to the console, and the Location’s address is added to the vector, accompanied by the appropriate string, “Up”, “Down”, “Left”, or “Right”. The player can choose a number from 1 to the size of the vector, which determines which Location address will be assigned to the current pointer. Once a move has been made, the new Location’s Event’s perform_event() function is called. If the result is -1, the game ends (this occurs when the Event is Prof and the Trainer object meets the requirements to end the game). If the result is 1, the Event was a Pokemon, and the current Location’s Event is swapped with the Event of a random Location in the blank array.

I haven’t thought about the AI class at all yet. I’ll have to see if I have time this week to figure it out. It would be easy to make it stupid, and just choose a random number between 1 and the size of the move_choices vector for each turn. Then maybe adding in prioritization of Poke-stops and Caves until sufficient numbers of pokeballs and megastones have been acquired, as well as avoidance of Professor Oak until the requirements have been met. The only challenge will be deciding how the AI will interact with the game. Will it only be able to parse through the output text, or will it have access to some internal variables, such as the number of elements in the move_choices vector? Unfortunately the answer is probably the former.

Testing Plan

The same get_pos_integer and get_nonneg_integer functions used for previous assignments were used for all user input, so logic errors are the focus, rather than input validation.

Things to check for:

- The user is not allowed to move off the grid of Locations.
- Pokemon evolve when the correct number have been caught.
- Pokemon that have already been caught are correctly identified, and their num_caught member is incremented, rather than creating a new element in the Trainer::pokemon array.
- When the user chooses to play again with the same map, the Trainer's items and Pokemon are reset, but the locations of the Events on the game grid retain their final values from the previous game.
- The Prof Event correctly identifies when the Trainer meets the requirements to end the game.
- When Pokemon flee or are caught, the Event is swapped with a Blank Event in another Location.
- The inventory is printed correctly.
- Only Stage 1 Pokemon can be evolved with a megastone.
- Debug mode tells the user the direction of each Event (including Professor Oak), but this information is hidden in normal mode.
- Movement around the grid functions correctly. The path taken does not impact the contents of a Location (other than scattering the Pokemon encountered along the way).
- etc…