Final Project Group B Proposal

SchillerQuest

Ephraim Benson, Anders Bruihler, David Higgs, & Will Schwarzer

In this project, we will create a top-down, turn-based role-playing game (RPG), in the style of Rogue[1] and NetHack,[2] a genre commonly referred to as Roguelike. In the game, the player will attempt to reach the bottom of a procedurally generated dungeon, to recover the bust of Schiller. Along the way, the player will fight monsters and find more powerful equipment, but there's a catch: when the player dies, they have to start over from the beginning. This ensures that each player choice is meaningful.

We recognize that this is an ambitious project. For that reason, our focus over the next few weeks will be first and foremost on creating a basic game that runs and plays as it should, with simpler features. We will then continue to add additional features (e.g. additional enemies, procedural generation, different weapons, etc.) as time permits, after we complete the base game.

Project Pieces

These are the features we plan on implementing in our game, in order of priority:

1. An ASCII display of the game world, as suggested in the sketches.
2. In a separate area of the GUI from the ASCII display, a user interface for holding other elements besides the game display. Some such elements include looking the player's inventory, opening an overall game menu, or player's stats.
3. Entities (one player and multiple AI-controlled monsters) occupying the game world.
4. Movement of entities through the game world so that only acceptable movements are possible (ie. no moving through walls or through active entities).
5. A turn system that allows for the player entity to move/act and then allow for non-player entities to respond and move.
6. Combat between entities. This can be understood as two entities having the option of interacting when in close enough proximity.
7. Handling user input which would allow a user to control an entity. The basics of this will allow the player to move and attack. Eventually, it should be able to handle other inputs in order to interact with items or non-enemy entities, if added.

[1] https://en.wikipedia.org/wiki/Rogue_(video_game)
[2] https://en.wikipedia.org/wiki/NetHack

8. Character and monster statistics can be added for more depth in gameplay and combat specifically. This would include having multiple monster types as well as status attributes for all entities such as health, mana and speed.
9. Items affecting player's statistics in order to add a sense of progress to the game. These items will affect entity statistics like combat, speed, or lighting. This will require "balancing" of statistics so that the game difficult is consistent.
10. Procedurally generated levels, with staircases between each level, and a boss or similar challenge on the last level.
11. A vision system for the player, such that the map starts out unexplored, and only becomes revealed as the player moves through the map. (In other words, the view only displays updates for those parts of the world that are within the player's line of sight.)

Finally, here is a list of other features we are considering but are unsure of the likelihood of completing. We would be happy if we implemented just a few of these.

- Different 'texture pack' options, including an emoji theme (no, really)
- Non-player-characters (NPCs), e.g. shopkeepers
- Magic
- Special non-boss level types (e.g. Sokoban[3])
- Environmental hazards (pitfall traps, etc.)
- A speed system, such that entities with a higher speed attribute can take more actions per turn
- Saving/loading a game (although the save will still be deleted upon player death)
- Achievements/stats (e.g. 100 zombies killed overall)
- Highscores list
- Easter eggs
- Multiplayer functionality
- Controller support
- In App Purchases (IAPs) for powerful weapons
- ~~Donations~~ Support-poor-college-students link or Patreon
- Put the game on Steam

The last few goals may be overly ambitious, but we enjoy a challenge.
(Yes, we're kidding.)

Design Patterns

Being a videogame, our project will use the MVC architecture. We will have a view, including a display of the game's ASCII art and any menu interfaces (such as for accessing the player's inventory). This view will, as specified for MVC, be the user's interface with the program. We will

[3] https://en.wikipedia.org/wiki/Sokoban

have a controller, which will handle user inputs. Some of these user inputs, such as opening the inventory screen, may be passed directly back to the view, while some others, such as making a move, will first be passed to the model. This model will handle the core of the game's logic, including player actions (once passed in by the controller), monster actions, generation of the game world, and so on.

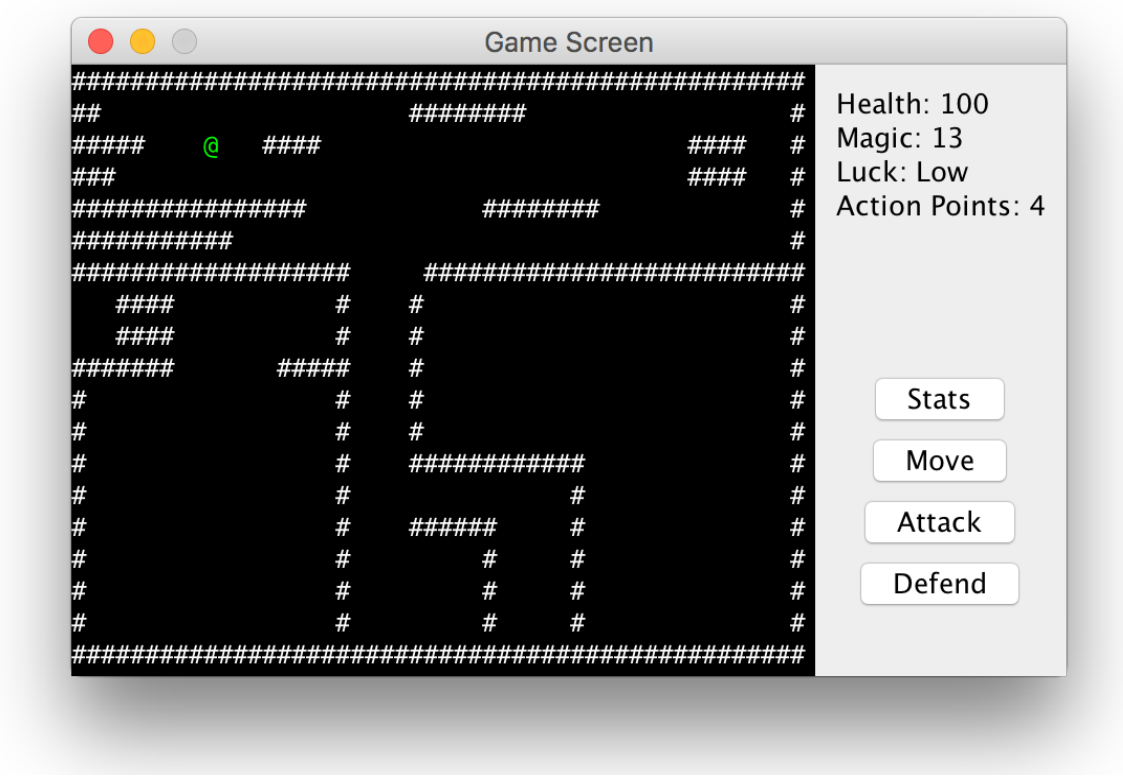
Of course, using MVC in turn implies various individual patterns that we'll use. We will have multiple possible states of the view - for example, if the user is looking at their inventory - and this means that we will need multiple versions of the controller. Therefore, the controller class will implement the strategy design. Certain in-game items will also use the strategy design pattern - for example, the player may have a dagger, sword, or other weapon; all of which will deal different damage amounts and in different ways, and the strategy pattern will allow these to be interchangeable.

The view and model classes will also implement the observer design, with the view as the observer and the model as the subject. The model will pass the view updated information about the game world whenever something changes (this will most likely be a push format for the observer pattern), so that the view can update the game's display.

Finally, the view will implement the composite pattern, as already shown in our GUI sketch: the main game window will have several sub-windows, with one for the world display, one for the player stats, one for the menu options, and so on. When, for example, the controller tells the view to update, it will only have to tell the highest-level view to update: each view will then take care of the rest by telling all its children views to update, until they reach the "leaves" of the window (i.e. the map, status text, etc.), at which point those atomic elements will in fact update themselves.

GUI Sketch

The following is a preliminary sketch of our GUI, created in Java with what will likely be the same elements that we eventually use to create the final GUI. In the sketch, the section on the left is the display of the game world. It is a top-down view: the @ is the player character, and the # are walls. The player menu is on the right. It displays the player's status, allows the user to interact with the game world, and may eventually display any notification text that pops up ("You hit the goblin.") during gameplay.



Team Roles

- Will will handle most of the AI in the game, including monster AI and (potentially) the player-vs-player AI for the competitive mode. At the start of the project, Will will also work on user input.
- Ephraim will handle drawing the game world with ASCII graphics and presenting the other buttons and labels in the UI. He will be focusing on the view portion of MVC.
- Anders will deal with level generation and making sure that the world is playable, starting with importing a world from a file and then modifying/randomizing as needed for each play.
- David will work on the design of the core game features. This will include creating classes for entities (and subclasses for players and monsters), items and (potentially) world features, and implementing the interactions between these classes.

As time goes on, the focus of the project will likely shift more towards game design and away from core logic. At that point, every project member will help out with design, and whatever polishing remains to be done.