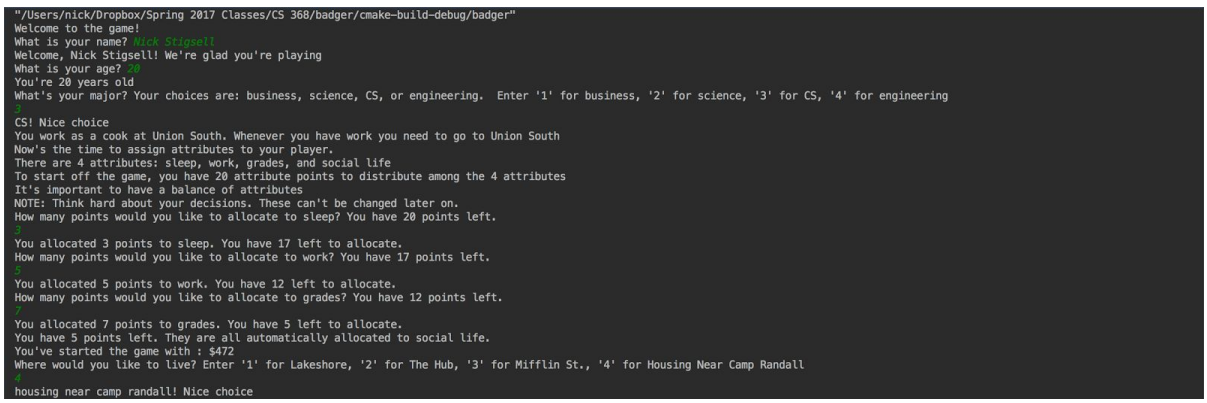
A Week in the Life of a Badger

Jared Hageny and Nick Stigsell

Overview

Our CS 368 final project is a text-based game called “A Week in the Life of a Badger” (AWLB for short). AWLB is a game that simulates what it’s like to be an undergraduate student at UW-Madison. The game begins by prompting the user to enter their name, major, living situation, and how much they value attributes such as grades, sleep, work, and social life. Based on which attribute is ranked the highest, the user is generated a class and work schedule for a week, along with money to start the game. For example, a user that ranks work as their highest attribute will be generated a schedule that involves more work than free-time, and this user will start with more money. Over the course of the week, the user is presented with situations that a typical student would face, and is asked to select one of many predetermined decisions such as study, relax, socialize, go out, etc. Throughout the game, the user will see that the decisions they make in each situation will cause them to gain or lose attribute points and money. For example, if the user has free time and decides to study, their grades attribute will increase, and if the user decides to socialize, their social attribute will increase. The game utilizes a graph to connect several campus locations that resemble the layout of UW-Madison’s campus. This will allow the player to roam around Madison as they please, while also forcing the user to get to certain points on the map to complete tasks such as work and class. For example, work can only be completed at Union South, and class can only be completed at the building that aligns with their chosen major. The game will be a text-based game played on the console, and will rely on user input via the keyboard to make decisions. As the name suggests, the game lasts one week, so after Sunday night, the game ends by printing the final player stats and thanking the player for playing. Just like there’s not a single “correct” way to go through college, there’s no “correct” way to play this game. If a player doesn’t value their education very much, they’re free to spend their free time socializing instead of studying. This puts the player in control and lets them make their own decisions.

Screenshots



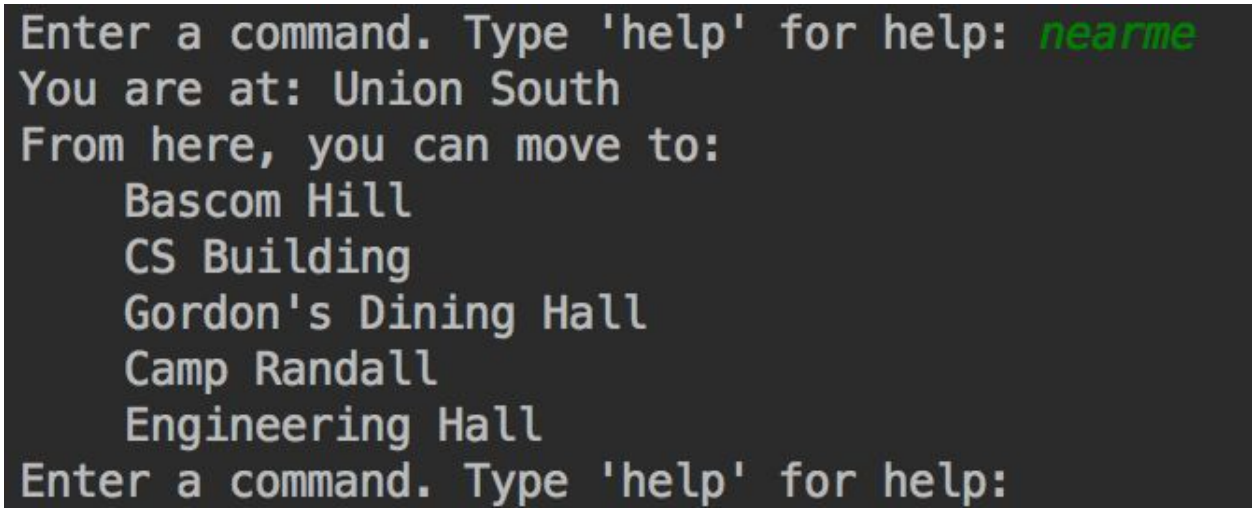
This shows the start of the game where the player attributes are set up.

```
Enter a command. Type 'help' for help: stats
Sleep: 3
Work: 5
Grades: 7
Social: 5
Bank balance: $472
Major: CS
Work location: Union South
Home: housing near camp randall
Class location: CS Building
```

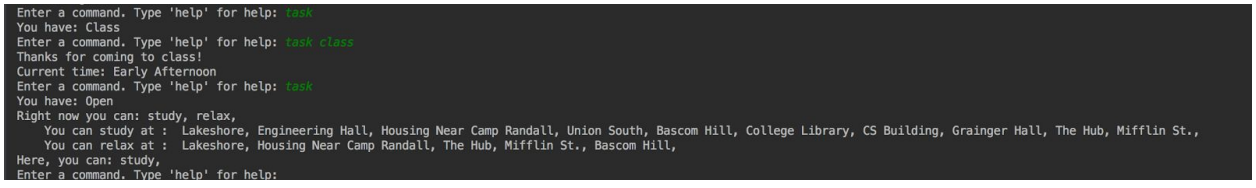
Time	Mon	Tues	Wed	Thur	Fri	Sat	Sun
Morning	Class	Work	Class	Open	Class	Open	Open
Early Afternoon	Open	Class	Open	Class	Class	Open	Open
Late Afternoon	Class	Open	Class	Open	Class	Open	Open
Dinner	Open	Open	Open	Open	Open	Open	Open
Night	Open	Open	Open	Work	Open	Open	Open

```
Enter a command. Type 'help' for help:
```

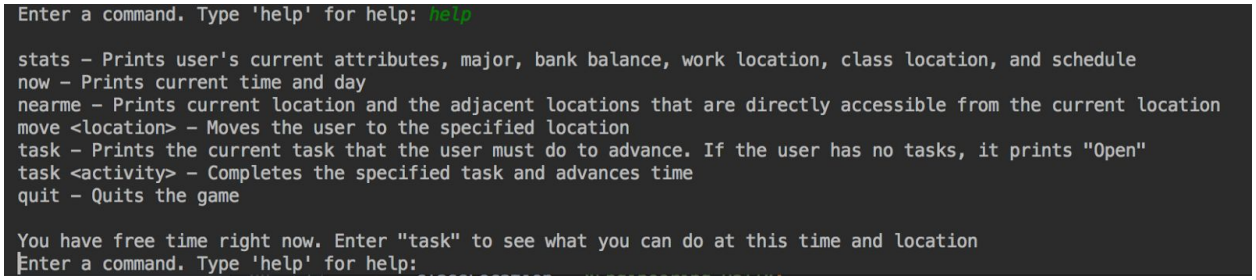
This shows the values of each attribute and the generated schedule. This screenshot shows a schedule based on grades being the highest attribute.



This shows the “nearme” command, which tells the player where they are and which locations are directly accessible from the current location.



This shows the “task” command, which tells the player what is currently on their schedule and what tasks are allowed at the current time and location.



This shows the “help” command, which provides the user with a description of each command that can be used during the game. The help command also has a wizard at the bottom that tells them what they should be doing right now. This helps confused players continue playing the game instead of becoming frustrated and quitting.

Implementation

The program is divided into the following classes:

Main - Controls execution of the program. Sets up the locations graph, player attributes, player schedule. This class has an infinite while loop function called processUserComands() that continuously gets user input and processes each command. Each command has a corresponding function that is called when that command is typed. For example, if the user types “stats”, then the processStatsCommand() function is called. The program is stopped by typing “quit”. The main class keeps track of 3 important global pointers so that each method in main can access them. Those pointers are: Player, Schedule, and Graph.

Graph - This class is where the locations are stored. Because the program has users walking around to different locations around campus, it made sense to store them in an undirected graph. The graph is implemented using a vector of locations, each of which stores an adjacency list. The graph is initialized in the setUpLocationsGraph() method in main.cpp

Location - This class represents a location in the game that a player can access. It stores a location name (both a capitalized display name and a lowercase “short name” for string comparisons are stored), a vector of allowable activities, and a vector of adjacent locations. The location objects are created in the setUpLocationsGraph() method in main.cpp and added to the graph.

Schedule - This class represents a player’s schedule throughout the game. There are 4 different schedules that can be assigned to a player. When the user inputs their 4 attributes at the start of the game, the highest attribute is used as the schedule. For example, if a user has the attributes (work, sleep, social, grades) = (3, 4, 4, 9), they would have a grades-heavy schedule. The schedule class also keeps track of the current day and time block during the game. This class also contains a few helper functions to reduce duplicate code. The schedule is created in the createSchedule() method in main.cpp

Player - This class represents a player in the game. A player object consists of a name, age, major, bank balance, attributes, home location, and current location. This class has a number of getter and setter methods to change attributes and bank balance.

The program does lots of string comparisons in order to make decisions, and that is one thing that I would refactor if I had more time to work on it. Overloading the == operator would make the code safer and easier to read, but unfortunately we ended up using a lot of code like the following: if(!string.compare("other string")). The compare() function in the string class returns 0 if the strings are equal and 1 otherwise, so the "!" is there to flip that bit for the if statement since in C++, 1 is true and 0 is false. In our initial plan, we were going to have some random events occur throughout the game, but due to time constraints, we had to cut that feature from our program. The game is still playable without that feature, it just serves to make the game seem more random and less structured. We feel that there's enough decision-making in the game currently where it's not a necessary feature right now.

Contributions

We began working on the game several months ago with 2-3 meetings to figure out the structure of the game. We felt that it was important to plan out as much as possible before coding so that by the time we started coding, there was less decision making and we could focus on implementation. We planned out the locations graph, schedules, decisions, attributes, in-game menu all before coding began. We even identified which classes we would need and which functions would be necessary within those classes. This planning made coding a very smooth process. We met in person several times and pair programmed most of the features. This ensured that each person was contributing and was aware of how the code was implemented. Jared focused on the Schedule class and the point system. Nick specialized in the Graph class and the in-game menu. The rest of the classes/features were worked on together.

Reflections

Throughout the course of this project, we learned a lot about how to build a large program from scratch. Through most computer science coursework, we are given a template and are told to implement certain parts of it, so it was interesting being able to start from scratch and add features one by one until the program was finished. Starting small and working our way up was the right way to build the game, rather than trying to implement everything at once. For example, we started with a simple function that got user input and stored it in a Player class. Once we had that working, we added the Locations graph, and eventually added Schedules and attribute points. Neither of us had much experience with using Git/Github, so it was fun to be able to learn that and use it for our program. It was a great tool that made sharing our code very easy! We also had never used the CLion IDE before, so we decided to both use it for our project. It's got some very advanced features, and we enjoyed learning about some of them. The built-in debugger was a very important feature that saved us from hours of manual debugging with print statements. Overall, working on this project was a great learning experience, and the lessons we learned should serve us well in future coding projects.