theResistance
Aleksandra Koroza, Rohan Ahammed, Shafali Gupta

APCS1 pd 08
2018-01-04

Final Project Proposal

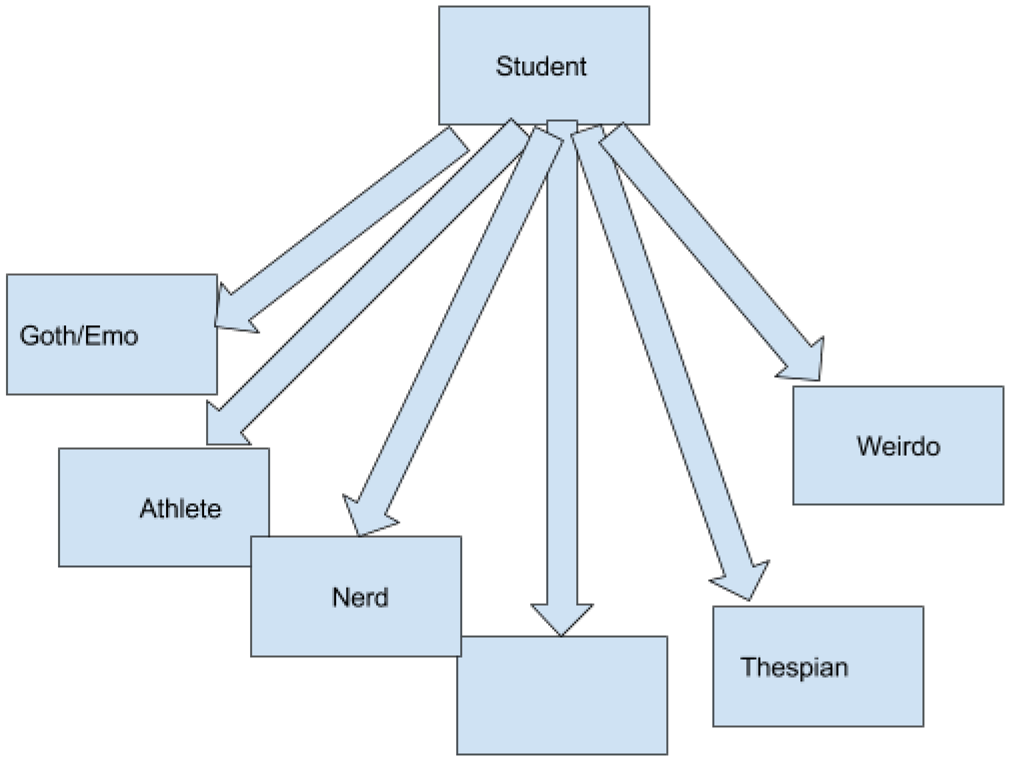
High School 2.0

We will endeavor to create a role-playing, text-based game in which we redefine social classes in the worst place in existence- high school. In the first stage of the game, each player is sorted into a certain “type” of person based on their reactions to a multiple-choice quiz. These “types” would be based off of typical high school characters as defined by pop culture (ie. Mean Girls; High School Musical; etc).

Types of Students:

- Athlete
- Nerd
- Thespian
- Goth/Emo
-
- Weirdo

Inheritance Tree:



- Uses the concepts of polymorphism and inheritance.
- Scenarios would be pre-determined and chosen randomly during game-play from an array that stores possibilities.

Student attributes:

- Friend Count
- Mental health
- Popularity (in their own sphere/ type of student)*
- Physical health
- Intelligence
- Score (calculated by taking into account each of the other attributes// will be used to compare to other players)*
- Social Status (this is used to determine the chances of being invited or accepted to certain social gatherings)

* Both score and popularity start at 0. Popularity (like other attributes) is either gained or lost based on the user's decisions. Score calculated when game-play ends.

Athlete:

- High physical health
- Low intelligence
- High friend count
- Average mental health
- High Social Status

Nerd:

- Low physical health
- High intelligence
- Average friend count
- High mental health
- Low Social Status

* nerd specific responses may impart high popularity points

Goth/Emo:

- Low physical health
- Average intelligence
- High friend count
- Low mental health
- Average social status

* goth/emo specific responses may impart high popularity points

Thespian:

- Average physical health
- Average intelligence
- High friend count
- Average mental health
- High Social Status

* it is harder to achieve popularity points as a thespian

Weirdo:
- Average physical health
- Average intelligence
- Low friend count
- Low mental health
- Low Social Status

* weirdo specific responses allow for dramatic climbs in all attributes

~~~~~~

After initializing the player's character, gameplay begins. The game would proceed over the course of 7 days (starting on a Sunday and ending on a Saturday of a regular school week).

- Turns would consist of the user responding to both hard-coded and randomly generated scenarios. The user's responses trigger methods that mutate attribute values based on the player's type of student. Some responses may also trigger mini-games. Scores resulting from mini-games may contribute popularity points.
- Game ends when a) mental health or physical health drop to 0 or b) end of 7th day reached. At end of gameplay, score is calculated using all of the attributes.

Possible scenarios (still under consideration):
- Party in midtown
- Sleepover
- Studying
- Choosing to go to school
- Talent shows
- Educational competition
- Relationships
- Bullying

Possible mini-games:
- Themed quizzes for each different category (eg. science quiz for nerd)
- Number guessing game for weirdo
~~~~~~

~~~~~~

Prioritized to do list:

- Create superclass Student, and subsequent (6) subclasses.
- Compose a driver class file that will be helpful in testing functionality of other classes.
- Work on developing core methods of Student and achieving functionality.
- Review previous homeworks (like YoRPG) in order to understand the structure of a driver class file.
- (Test incrementally.) ← important at all stages!
- After achieving a working version, look to stretch with work with csv files and file reading.

Timeline:

(~2 days)

1) Review driver files, and solidify understanding on abstract methods and polymorphism.
2) Set up working class files for all super/sub classes and Woo.java.

(~4 days)

3) Work on defining abstract methods in class Student.
4) Work on driver file to be able to test newly written methods.
5) Start working on each superclass at a time. Creative work to develop scenarios and responses; how each response affects

(~2 days)

6) Incorporate mini-games/ aspects to make user experience more entertaining

(~4 days until submission)

7) Finish driver file
8) Robustify code.
9) Explore csv files and file reading.
10) Implement high score storage system.
11) Test, test, test.
~~~~~~