CS 202 Fall 2022 - Extra Credit Assignment

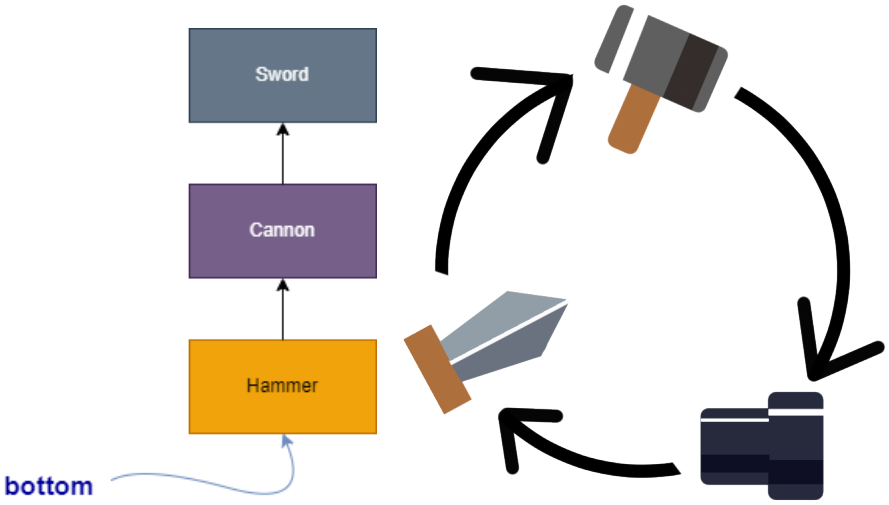
Linked Lists and Recursion Practice

Overview

Glory is won in the heart of battle. Let's get some bots battling. In this assignment, you will be simulating bot battles between two competitors. Each robot will be made of segments stacked on top of each other to form an entire BattleBot. They will then compete rock-paper-scissors style in order to find out who is the champion. Each robot is made of some number of segments each with some attached weapon: A hammer, a sword, or a cannon. Hammer segments will beat cannon segments which will beat sword segments, which will beat hammers, forming a weakness triangle.

For our battles, each robot will be pitted against another in a series of rounds. For each round, segments at the same height in both robots will compete and win depending on the previously mentioned weakness triangle. If a segment loses a battle, it will be destroyed and all segments above it will fall down due to gravity. Once a bot has lost all of its segments, it loses the battle and its opponent is crowned the winner.

Each bot will essentially represent a linked list, with the head being the bottom of the bot and the segments above it linked on top. We will have two major classes: a BotSegment base class that will be inherited by each weapon type, along with a BattleBot class to represent a bot with a number of segments. You will be tasked with writing some of the functions to assemble, display, and let each bot battle. For a better graphic example of fights, please see the accompanying video found here: https://youtu.be/cnsAptYpIlY



UML Diagrams

BotSegment
- type : SegmentType + segmentAbove : BotSegment*
getSegmentType() : SegmentType + compareSegments(BotSegment*, BotSegment*) : static int + printSegment() : virtual void + BotSegment(SegmentType) + BotSegment(BotSegment*, SegmentType) + ~BotSegment()

HammerSegment, SwordSegment, CannonSegment : BotSegment
+ printSegment() : void + HammerSegment(), SwordSegment(), CannonSegment() + HammerSegment(BotSegment*), SwordSegment(BotSegment*), CannonSegment(BotSegment*) + ~HammerSegment(), ~SwordSegment(), ~CannonSegment()

BattleBot
+ bottom : BotSegment*
+ playBattleRound(BattleBot&, BattleBot&) : static BattleStatus + printBot() : void + addSegment(BotSegment*) : void + hasNoSegments() : bool + BattleBot() + ~BattleBot()

Classes and Functions

Below is a list of functions and variables that are important for this assignment. ***Variables are in green***, **functions that you will need to write are in red**, **functions implemented for you already are in**

blue, and **functions that are abstract that will be implemented later are in magenta**.

Enumerations

- **SegmentType {SWORD, CANNON, HAMMER}** - Enumerations for the three types of weapons each segment could use. Used to easily identify what kind of part a segment is without using a cast.
- **BattleStatus {BOT1_WIN, BOT2_WIN, DRAW, NO_RESULT}** - Enumerations for the status of the battle after a round. This tells whether either bot has won, there is a stalemate/draw, or that no result has been reached yet, so the game can continue being played.

BotSegment

The abstract base classes for the parts that make up a battle bot. Each of these segments will be stacked one on top of the other to form an entire bot, each with a unique weapon. This class and its derived classes have been implemented for you already.

- ***SegmentType type*** - The type of weapon that this segment uses
- ***BotSegment* segmentAbove*** - A pointer to the segment stacked on top of this one within a bot. This will be nullptr if there are no segments above this one (i.e. this segment is the top)
- **SegmentType getSegmentType()** - Getter for the ***type*** variable
- **static int compareSegments(BotSegment* segment1, BotSegment* segment2)** - Compares the two segments and determines the result of their battle using the weakness cycle. If segment1 beats segment2, this function returns 1, if segment1 loses to segment2, this function returns -1, and if neither one wins it returns 0. The result can be used in the BattleBot's **playBattleRound** function to determine what segments win what fights.
- **virtual void printSegment()** - Prints the segment to cout along with its name.
- **BotSegment(SegmentType type)** - Sets the segment's ***type*** and then makes it point to nullptr for its ***segmentAbove***
- **BotSegment(BotSegment* segment_above, SegmentType type)** - Initializes the class variable members to the corresponding parameters. Will likely be unused in your code, but exists for convenience
- **virtual ~BotSegment** - Virtual destructor that does nothing. Exists so that the other destructors can execute correctly

HammerSegment, SwordSegment, and CannonSegment

These three class inherit from the BotSegment and have similar functions, but each have a different weapon. As such, they have been condensed to the one section

- ***void printSegment()*** - Prints the segment to cout. This will display some special character for the segment along with its name. See the sample output for a better visual
- **HammerSegment()** - A default constructor for all three types of segments. Sets the ***type*** of the segment to match the segment's type via the base class constructor
- **HammerSegment(BotSegment* segment_above)** - Sets the segment's ***type*** to the corresponding type and then sets the segment_above, both via a base class constructor
- **~HammerSegment()** - Destructor that does nothing, but exists to allow each segment to be deleted gracefully

BattleBot

This class represents a fully built BattleBot that will participate in fights. Each bot is made up of a number of segments.

- ***BotSegment* bottom*** - A pointer to the bottom-most segment in the bot. This will be nullptr if there are no segments in the bot (i.e. it is unitialized or destroyed)
- **void printNextSegment(BotSegment* segment)** - Recursive function that attempts to print the next segment above the given one assuming it exists, and then prints the given. If no segment does exist above it, then do not recurse. Another framing of this function is that it should print the bot where the given segment is the bottom.
- **static BattleStatus playBattleRound(BattleBot& bot1, BattleBot& bot2)** - Plays a single round of the BattleBot game. This will be the most extensive function. This should compare all corresponding parts of each bot and let them battle. That is, the two segments on the bottom of each bot should battle, then the ones above the bottoms should battle, and this should continue until the top of at least one of the bots is reached. Whenever a segment loses a battle, it should be removed from its bot and the relevant pointers should be updated. If a segment is not the bottom, the segment below it should now be pointing at the segment above the one that was deleted, else if the bottom is destroyed, update the bottom pointer. It is recommended to do this function using a while loop and the provided pointers in the skeleton code, along with the **compareSegments** function. This function then returns a BattleStatus enum on the results of playing the single round. Please see the linked video for a graphical demonstration of this function.
- **void printBot()** - This should print the entire bot. This functions serves simply to act as a wrapper and make the initial kick-off call to the **printNextSegment** function starting at the bottom of the bot.
- **void addSegment(BotSegment* segment)** - Adds a segment to the very top of the bot. Like a standard linked list implementation, this should start at the head/bottom, iterate until it finds the last segment in the bot, and then attached the segment to the end. If there is no bottom set yet, the segment should become the bottom.
- **bool hasNoSegments()** - Tells if the bot has no segments and is empty. Simply returns true if the bottom is null. This can be used to tell if a bot has been destroyed completely in **playBattleRound**
- **BattleBot()** - Initializes the Bot to have no segments (i.e. makes the bottom point to no segment)
- **~BattleBot()** - Destroys all remaining segments in the bot, starting at the bottom.

Global Functions

Below are global functions used only by main, as well as a brief description of main for your testing purposes

- **void printResult(BattleStatus result_status)** - Prints the results of a single round between BattleBots in main.
- **void simulateBattle(BattleBot& bot1, BattleBot& bot2)** - Simulates a battle between BattleBots, printing information between each round.
- **int main()** - Provides 4 test cases and simulates each. The first test case will show whether addSegment and printing work, with the final 3 test cases being for bot 1 winning, bot 2 winning, and a draw, respectively.

Compiling / TO-DO

Use the provided makefile to compile the program. You can build simply with the bash command **make**. This will output an executable called battle_sim.

Only the BattleBot class is incomplete, so you need only work within BattleBot.cpp. It is suggested to start with the **printNextSegment** and then the **addSegment** function so that they can be used with the first test case and for debugging purposes. Then, move onto the BattleBot destructor and the **playBattleRound** function. The playBattleRound function will likely be the most in-depth program and the bulk of the program.

Sample Output

Case 1: Testing Printing

```
Run which test case?
(1) Test printing and adding segments
(2) Test bot1 wins
(3) Test bot2 wins
(4) Test draw
1
xxxxxxxxxxxxxxxxxxx
x       SWORD      x
xxxxxxxxxxxxxxxxxxx
oooooooooooooooooo
o      CANNON      o
oooooooooooooooooo
oooooooooooooooooo
o      CANNON      o
oooooooooooooooooo
********************
*      HAMMER      *
********************
```

Case 2: Bot 1 Wins

```
Run which test case?
(1) Test printing and adding segments
(2) Test bot1 wins
(3) Test bot2 wins
(4) Test draw
2
----------- Initial bots -----------
Bot1:
oooooooooooooooooo
o      CANNON      o
oooooooooooooooooo
xxxxxxxxxxxxxxxxxxx
x       SWORD      x
xxxxxxxxxxxxxxxxxxx
********************
*      HAMMER      *
********************

Bot2:
oooooooooooooooooo
o      CANNON      o
oooooooooooooooooo
oooooooooooooooooo
o      CANNON      o
oooooooooooooooooo
oooooooooooooooooo
o      CANNON      o
oooooooooooooooooo

----------- Round 0-----------
Bot1:
```

```
oooooooooooooooooo
o      CANNON      o
oooooooooooooooooo
********************
*      HAMMER      *
********************
Bot2:
oooooooooooooooooo
o      CANNON      o
oooooooooooooooooo
oooooooooooooooooo
o      CANNON      o
oooooooooooooooooo

----------- Round 1-----------
Bot1:
oooooooooooooooooo
o      CANNON      o
oooooooooooooooooo
********************
*      HAMMER      *
********************
Bot2:
oooooooooooooooooo
o      CANNON      o
oooooooooooooooooo
Bot 1 wins!
```

Case 3: Bot 2 Wins

```
Run which test case?
(1) Test printing and adding segments
(2) Test bot1 wins
(3) Test bot2 wins
(4) Test draw
3
----------- Initial bots -----------
Bot1:
oooooooooooooooooo
o      CANNON      o
oooooooooooooooooo
xxxxxxxxxxxxxxxxxxx
x       SWORD      x
xxxxxxxxxxxxxxxxxxx
********************
*      HAMMER      *
********************

Bot2:
********************
*      HAMMER      *
********************
oooooooooooooooooo
o      CANNON      o
oooooooooooooooooo
```

```
xxxxxxxxxxxxxxxxxxx
x       SWORD      x
xxxxxxxxxxxxxxxxxxx
Bot 2 wins!
```

Case 4: Draw / Stalemate

```
Run which test case?
(1) Test printing and adding segments
(2) Test bot1 wins
(3) Test bot2 wins
(4) Test draw
4
----------- Initial bots -----------
Bot1:
ooooooooooooooooooo
o      CANNON      o
ooooooooooooooooooo
xxxxxxxxxxxxxxxxxxx
x       SWORD      x
xxxxxxxxxxxxxxxxxxx
**********************
*      HAMMER      *
**********************

Bot2:
**********************
*      HAMMER      *
**********************
**********************
*      HAMMER      *
**********************
xxxxxxxxxxxxxxxxxxx
x       SWORD      x
xxxxxxxxxxxxxxxxxxx
ooooooooooooooooooo
o      CANNON      o
ooooooooooooooooooo

----------- Round 0-----------
Bot1:
xxxxxxxxxxxxxxxxxxx
x       SWORD      x
xxxxxxxxxxxxxxxxxxx
**********************
*      HAMMER      *
**********************
Bot2:
**********************
*      HAMMER      *
**********************
**********************
*      HAMMER      *
**********************
xxxxxxxxxxxxxxxxxxx
```

```
x       SWORD      x
xxxxxxxxxxxxxxxxxxxx

----------- Round 1-----------
Bot1:
xxxxxxxxxxxxxxxxxxxx
x       SWORD      x
xxxxxxxxxxxxxxxxxxxx
Bot2:
********************
*      HAMMER      *
********************
xxxxxxxxxxxxxxxxxxxx
x       SWORD      x
xxxxxxxxxxxxxxxxxxxx
It's a draw!
```