California State University, Sacramento
College of Engineering and Computer Science

Computer Science 35: Introduction to Computer Architecture

Fall 2022 – Project – *Wizard Battle*

Revision 1

Overview

Your first semester at ***Hogwarts School of Witchcraft and Wizardry*** is going incredibly well. At least so far….

One of your favorite classes is Charms taught by the Ravenclaw Head of House – tiny little Professor Flitwick.

"Okay Ravenclaws", squeaks Professor Flitwick, "we are going to practice casting charms, hexes, and jinxes on each other."

"Oh, like a dueling club.", Joe Gunchy, the Prefect, says in a rather confident voice.

Flitwick shakes his head. "Nope. This is more of a freeform version. It's much like the muggle game Bumbler Cars!"

"What's that? Bumper… what? A car… like a train?", stammers a talk dark-haired girl you think is named Ada.

Flitwick frowns. "No, no… How do I explain? Hmmm… Okay, the best way for you non-muggleborns to understand it's like a snowball fight."

"Oooooooo. 'Chaos' in other words!" smiles a tall red-headed 6-year with a mischievous grin. She claps her hands together and rubs them ominously.

Flitwick nobs agreeably. "Yes, everyone please attack everyone else. 'Attack' is too strong a word. Go after. Yes, go after each other. Nothing horrible, mind you. Just your basic hexes and jinxes. Got it? Okay… now, begin".

The students spread out around the walls and begin to cast spells, quite benignly, at each other. Students are smiling, laughing, and chiding each other with comical taunts. You get hit by a Hair Raising Hex which gives you the appearance of a fluffy dog. In return, you hit your attacker with a Sneezing Hex that causes them to sneeze, fly into the air, and land on a desk – breaking several bottles of ink.

But, as the minutes pass by, things quickly start to take a turn for the worse. The students, at first playing, are getting increasingly aggressive. So, in a mere couple of minutes, books and furniture are flying around the room. The classroom windows are soon all shattered.

"Enough! Stop! Stop! STOP!" yells poor little Professor Flitwick as a misfired jinx sends him flying out a shattered window. You can hear continuing to yell "stop!" as his voice fades into the distance.

Your Task

Time to make a game – a game that will fully show of your knowledge of assembly and your immersion into nerd culture!

Okay, the second part might not be true, but you are going to use your skills to make a game similar to Advanced Dungeon and Dragons – well a Harry Potter version of it. Long before MMORPGs, players would sit around a board and play the game in person. The game itself is rather simple.

Each player has a number of attributes that affect how strong, intelligent, lucky, etc… they are. They game uses dice – and probability – to determine if something happens. Players use their knowledge and skill to increase them in their favor. Add to the mix a fantasy world, dungeons, dragons, and gnomes – and you have a fun adventure! Fortunately, you don't have to implement the whole complexity of the game. You will only write a small subset of what the real game had.

Sample Output

Your solution doesn't have to look exactly like this. However, this should show you the basic gameplay.

For readability, the user's input is displayed in **blue** and the random number is displayed in **red**. You don't have to use color (unless you are going for extra credit). *As always, please feel free to change the wording of the text.*

```
Welcome to the Wizard Battle!
Written by Joe Gunchy

How many players? 3

PLAYER 0
Health: 100
Your target: 1
Spell blasts them for 13 points

PLAYER 1
Health: 87
Your target: 0
Spell blasts them for 2 points

PLAYER 2
Health: 100
Your target: 0
Spell blasts them for 7 points

PLAYER 0
Health: 93
Your target: 2
Spell blasts them for 16 points

...
```

Restarted at 0

Pseudocode

The following is the basic logic of the game in Visual Basic-like pseudocode.

```
Display the title of your program
Display name
Ask how many players.
Input the number of players
Set the current player to 0
Loop
    (The player attacks… if they are alive)
    If the current player is alive (health[current player] ≥ 1)
        Display the player # and their health
        Ask them for their target
        Compute the points to subtract (use a random value).
        Display a message to the screen telling the user they hit for #points
        Update health[target]. (If you damaged the target, subtract from this table entry.)
    End If

    (Time to count the surviving players)
    Set the number of surviving players to 0
    Loop and check each of the number of players
        If (health[check] ≥ 1) then increment the number of surviving players.
    End Loop
    If the surviving players ≥ 1 then exit

    (Next player's turn)
    Increment the current player to the next one. Make sure to go back to zero if needed.
End Loop
```

Gameplay

The game is a "free-for-all" between different players. Each player takes turns attacking other players. The last surviving player wins!

The following is the basic logic of the game:

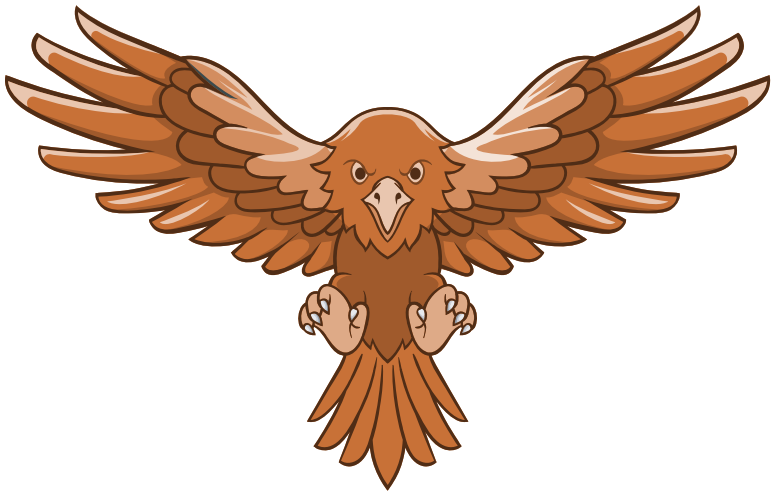
1. Input the number of players
2. The game loops until only one player survives
    a. Player selects a target player
    b. The game rolls "dice" and deducts the points from the target's health
    c. Go to the next player

Have Fun

Use your imagination. Your game doesn't have to be Dungeons and Dragons related. You can base your game on a fun theme that you want.

For example, here are some possible battle scenarios:

- Kittens
- Cartoon: Spongebob Squarepants, Rick and Morty, Archer, Powerpuff Girls, etc.…
- Politics
- Movie: comedy, sci-fi, horror, etc…
- A video game
- Television program
- Characters from a book
- etc…



Tips

Create a Health Table

Use the "target" number to index it.

Write you program in parts

DO NOT attempt to write the entire program at one time. If you do, you won't be able to debug it. Experienced programmers use incremental design. Make a basic program and, very slowly, add the features you need.

For example, in this project, don't initially worry about exiting the loop (if the number of surviving players is 1). You can code this last. First, try to cycle through the players in an infinite loop. You can press Control+C to exit any UNIX program.

Values you need to save / calculate

You need to store the following values (at a minimum) to get your project to work. You can store this information in either other registers or memory.

- Total **number of players**
- Array that stores the **health** of each player
- **Current player**
- Selected **target**
- Something to hold the **points** you randomly create for damage (for healing if you do extra credit)). Use this to display to the screen and change the health.
- The **surviving characters**

Random Numbers

The library has a built-in subroutine called "Random" that you must use to make your project work. Please read the documentation on how to use it. Cycling Players

Extra Credit

1. ***Input validation - 5 points***

   It is possible for the user to enter an invalid target index (i.e. less than zero or larger than the number of players).

2. ***Who won? – 5 points***

   When the program is complete, display to the screen which player won the battle.

3. ***Color – 5 points***

   Make use of color to enhance your game. The color must be meaningful – don't just set the color at the beginning of the program.

4. ***Player has multiple "spells" – 5 points each for a max of 15.***

   Create additional spells/moves. To make the game more exciting, each player could have an assortment of spells and attacks available. Perhaps you can make a spell that heals or makes a player skip a turn. Naturally, the attacks that do the most damage should be more difficult to land. Use your imagination. Have fun!

   Here is one possible example:

```
PLAYER 1
Health: 88
Your target: 0
1. Cast Deterioration Hex
2. Cast Pepper Breath Jinx
3. Sneeze
Your choice: 2
...
```

5. ***The attack can miss – 10 points***

   Not every attack – whether it is a spell cast, a cat's claw, etc… will necessarily hit the target. For this extra credit, make it possible that an attack will fail. This is the equivalent of rolling dice.

6. ***Player's Character Name – 5 points***

   To give the game a richer theme, display the character's name along with their number. You won't input this this a keyboard. Rather these names will be read from a table – much like your Vending Machine lab.

   For example, you could have this output:

```
PLAYER 0: Harry Potter
Health: 100
Your target: 1
Spell blasts them for 13 points

PLAYER 1: Draco Malfoy
Health: 87
Your target: 0
Spell blasts them for 2 points
```

7. ASCII Art – 5 point each for a max of 15.

Use ASCII-art to make your program exciting. The ASCII-art must be meaningful and not something overly simple like:

```
==========:)
```

It's a happy worm!

You can use multiple **`.ascii`** directives under the same label. Only place the **`\0`** at the end of the final one. You will also have to put a backslash **`\`** before any other backslashes or double-quotes **`"`**. For example, the following creates a square.

```
Square:
    .ascii "****\n"
    .ascii "*  *\n"
    .ascii "****\n\0"
```

Requirements

YOU MUST DO YOUR OWN WORK. DO NOT ASK OTHER STUDENTS FOR HELP.

If you ask for help, both you and the student who helped you will receive a 0. Any student using Discord will receive a 0. Based on the severity, I might have to go to the University.

1. Create a nice name for your game.
2. Display your name on the screen when the program starts. You are the author - you should get credit!
3. Ask the number of players as the game begins. You need to support at least 10. So, your table should have 10 consecutive **`.quad`** directives.
4. Ask each player for a target. Randomly damage the target player. Display now many points they lost. The exact numbers are up to you.
5. Cycle through the players
6. Loop until only one player remains.
7. Comment your code!
8. Proper formatting: Labels are never indented. Instructions are indented. Add blank lines for readability.

Due Date

The assignment is due in three weeks on **December 9th**. I strongly suggest that you get to work on this assignment as early as possible. If you did well on your labs, it shouldn't take more than 2 hours.

Connecting to the Server from Home

Step 1 – Connecting to the VPN

To get access to the server, which we will use for our labs, you must connect to the GlobalProtect VPN. There are instructions on our website.

Step 2 – For Windows

If you are using Windows, you need to download and install a copy of MobaXterm or PuTTY. Once you have installed it, open the application, and connect to the following address using SSH (Secure Shell).

```
coding1.ecs.csus.edu
```

Step 2 – For Macintosh

Open the Terminal program. This is the same UNIX prompt that you get when you connect to Athena. Mac-OS X is a version of UNIX. Neat! Once at the prompt, type the following where ***username*** is your ECS username. You might have to manually type "yes".

```
ssh username@coding1.ecs.csus.edu
```

Step 3 – Logging In

Once you are connected, you will be given the standard UNIX prompt:

1. Enter your username. You don't have to enter your entire e-mail address. Just enter the part before the @.

2. Enter your password. When you are on-campus, this part is normally skipped. However, when off-campus, you have to enter it. Note: UNIX doesn't echo characters when entering a password. You won't see anything visual on the screen, but UNIX is listening. Hit enter when done.

Getting the Course Library (csc35.o)

This course uses a library object file. The library contains a large number of utility functions that will allow you to easily print integers, strings, and other useful tasks. But, to use the library, you first need to obtain it.

You need to use a UNIX command called "curl" (Copy from URL). Please type the following at the command-line and press the Enter Key. It will download the library from my website and save it to your account. **Type the following verbatim!**

```
curl devincook.com/csc/35/csc35.o > csc35.o
```

UNIX will display information during the download. You can ignore this. It means it worked correctly.

```
[dcook@ecs-pa-coding1 ~]$ curl devincook.com/csc/35/csc35.o > csc35.o
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  6624  100  6624    0     0   5434      0  0:00:01  0:00:01 --:--:--  5433
```

You can check that the file downloaded by typing **`ls`** and pressing the Enter Key. You should see the library file listed.

Submitting Your Project

This project may only be submitted in Intel Format. Using AT&T format will result in a zero.

To submit your lab, you must run Alpine by typing the following, and, then, enter your username and password.

```
alpine
```

Please send an e-mail to yourself (on your Outlook, Google account) to check if Alpine is working. To submit your project, send the assembly file (do not send the a.out or the object file). Send the .asm file to:

```
dcook@csus.edu
```

UNIX Commands

Editing

Action	Command	Notes
Edit File	**nano** *filename*	"Nano" is an easy-to-use text editor.
E-Mail	**alpine**	"Alpine" is text-based e-mail application. You will e-mail your assignments it.
Assemble File	**as -o** *object source*	Don't mix up the *object* and *source* fields. It will destroy your program!
Link File	**ld -o** *exe object(s)*	Link and create an executable file from one (or more) object files

Folder Navigation

Action	Command	Description
Change current folder	**cd** *foldername*	"Changes Directory"
Go to parent folder	**cd ..**	Think of it as the "back button".
Show current folder	**pwd**	Gives the current a file path
List files	**ls**	Lists the files in current directory.

File Organization

Action	Command	Description
Create folder	**mkdir** *foldername*	Folders are called directories in UNIX.
Copy file	**cp** *oldfile newfile*	Make a copy of an existing file
Move file	**mv** *filename foldername*	Moves a file to a destination folder
Rename file	**mv** *oldname newname*	Note: same command as "move".
Delete file	**rm** *filename*	Remove (delete) a file. There is **no** undo.