CS 162 Assignment 3 Report

Kelvin Watson
OSU ID 932540242

Requirements

In this assignment, I am being asked to design and build a fantasy combat program by creating character objects and explore polymorphism. More specifically, I am being asked to:

- perform object-oriented analysis and object-oriented design prior to writing any code. In particular, I need to
  - determine which data members are needed
  - determine which member functions are required for the parent (Humanoid) class and subclasses
  - *ensure that I create an appropriate copy constructor, overloaded assignment operator and destructor in addition to a constructor if pointers are required*
- instantiate objects of derived classes of the parent Humanoid class using pointers to the Humanoid class. e.g. `Humanoid* ptr = new Goblin();`
- run a fantasy combat scenario with different characters using a driver program such as main()
- understand how to work with an abstract class

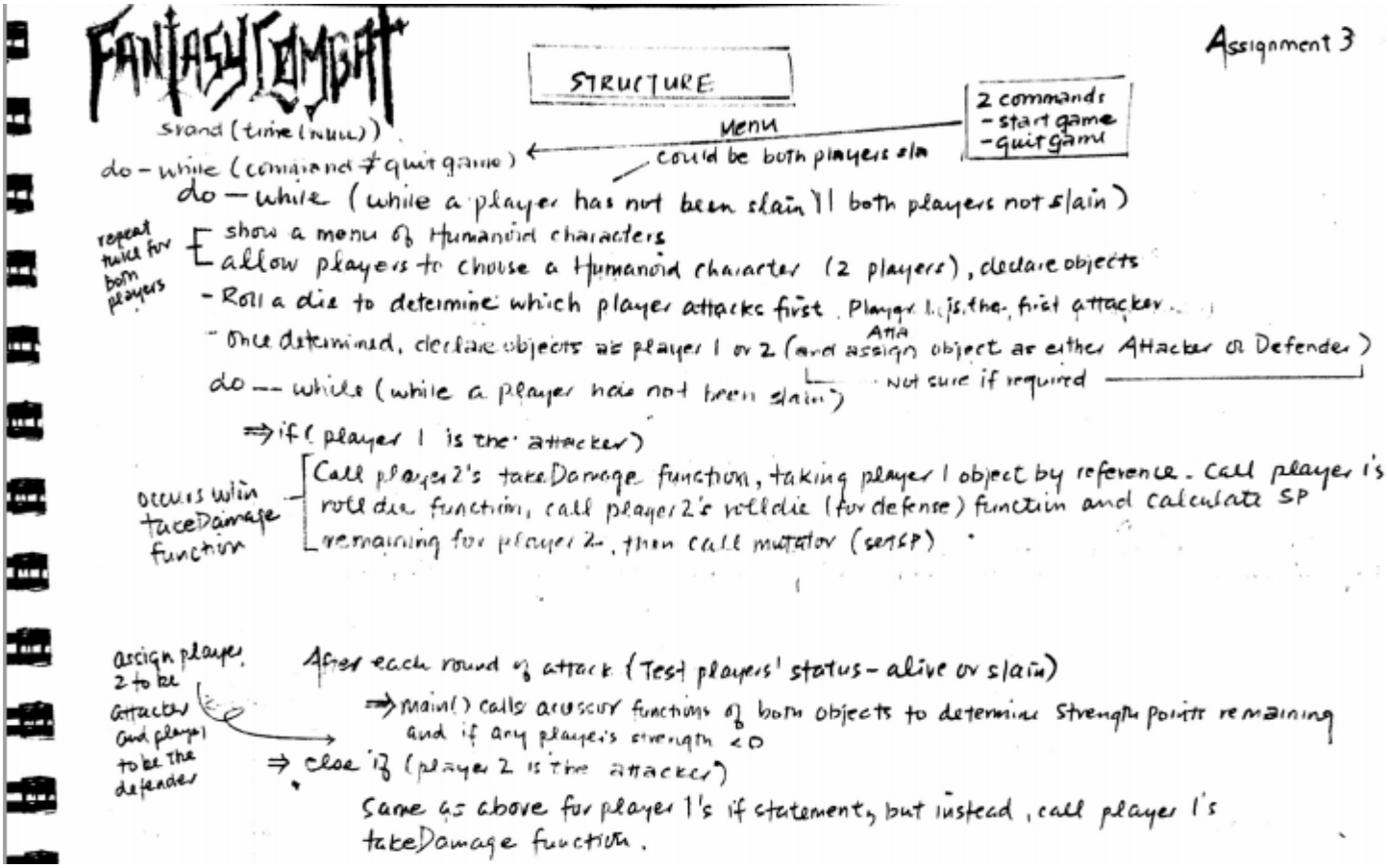
Assumptions:

- I assumed that this game consist of two players, i.e. one-on-one combat.
- I used mostly double (floating point) numbers because attack values can be halved by the Goblin's chance of cutting its opponent's Achilles tendon
- Since attack, defense, armor are calculated values that depend on other data members, they should be calculated and returned, instead of saved as attributes to avoid stale data[1].
- As the game may be further developed in future assignments, I named my character class Humanoid to be more precise. I looked up each character to make sure they were indeed Humanoid in nature.
- I assumed I needed to create another driver function (driver.cpp) in addition to the game program (assignment3.cpp) to test my program.

Uncertainties

- I am unsure if static member variables or static member functions will be useful for this assignment.

Design (continues onto next 2 pages, includes program structure and hierarchy)



Hierarchy (Inheritance)

→ Notes: Since Character class will not be instantiated, it should be an abstract function w/ pure virtual functions

As it seems we may be developing the game further in later assignments, I called the class

Since achilles can halve the attack of non-Goblins, should do all arithmetic in floating point.

Uncertainties

- one-on-one combat? desired to

Lessons learned void xyz (Base * ptr) only This can only access protected members not Base * arguments!

Assumptions

- use doubles as the data type because non-goblins' attacks can be halved.

firstTurn?

Since attack, defense, and armor are calculated values, that depend on other member variables, they should be calculated and returned by member functions instead of saved as attributes, to avoid stale data!. These functions should Δ the strength directly.

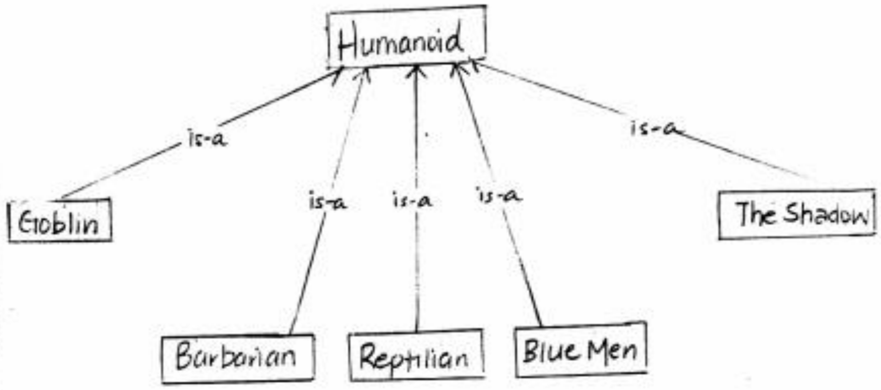
(Which one is pure virtual? — would have to be redefined in all classes)

same for almost all (except Goblin, so only Goblin and shadow modifies Achilles static int of other classes)

Achilles: call an function (set) in the opponent to alter its static double 0.5 Achilles

means all characters should have a static double called Achilles that is initially set to 1

Lessons – virtual is not always overridden (redefined) but offers the ability to redefine !. option exists to use the base class version! (no redefine needed)
- static int can be defined in class ‼ ☺



enum Role { ATTACKER, DEFENDER }

HUMANOID — BASE CLASS (ABSTRACT)

armor : double (since attack/damage can be halved, use doubles)
strength : double
~~achilles : bool~~
achilles : int
player : Role
playerNumber : int

Member Functions
+ sixSidedDef() double virtual =0 (Pure Virtual — all classes MUST use 6-sided die for defense)
+ attackOpponent() double virtual =0
+ tenSidedAttack() double virtual
+ setRole
+ getStrength
+ takeDamage
+ roll6Die()
+ roll10Die()
+

For the different characters, incorporate phrases such as "you dodged an attack" for larger damage or evaded attacks to indicate action taken and add to the story

Reptilian

+ sixSidedDefense()
+ attackOpponent()

Goblin

+ sixSidedDef() : virtual double
+ attackOpponent() : virtual double
+

Barbarian

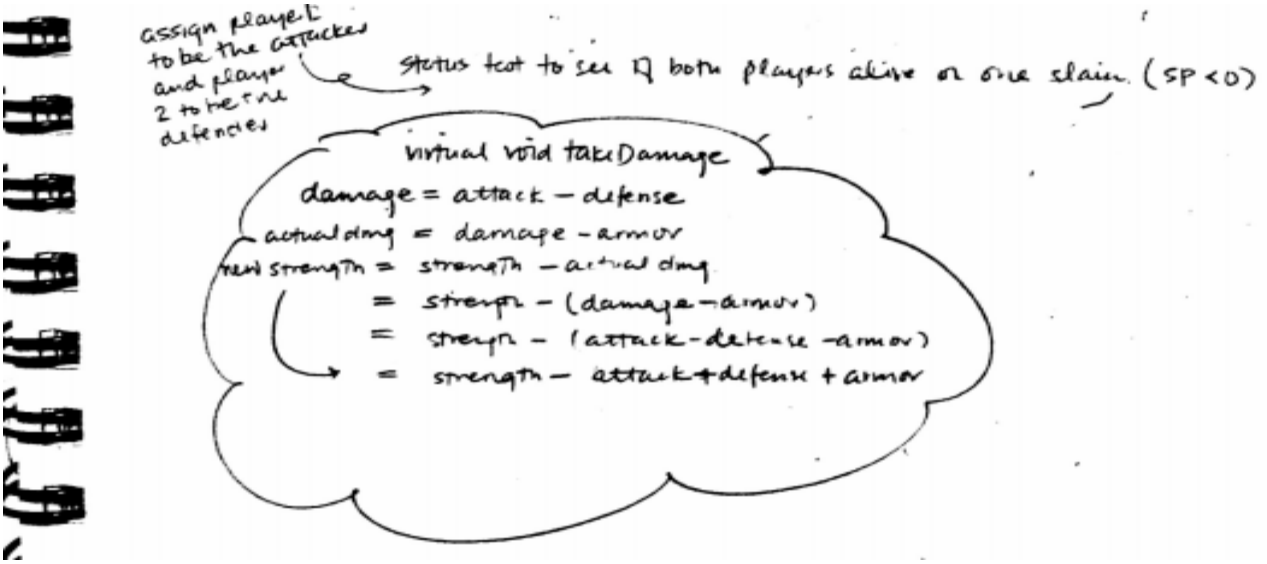
+ sixSidedDef()
+ sixSidedAttack()
+

Blue Men

+ attackOpponent
+ sixSidedDefense

The Shadow

+ sixSidedDefense
→ 50% chance of no damage
random = rand() % 2;
if (random == 1)
No damage
else
as per usual sixSidedDefense function
+ attackOpponent



Redesign and revisions: Regarding the overall structure, my program did not require too much redesigning or rethinking. I spent several hours designing to try and capture the smaller details. However, after I started coding, I realized there were important details that I had missed or had gotten incorrect on my design. For instance, my design specified a static int called achilles for every derived class, which would serve as a factor multiplied to every attack. I later realized that a static int would not make sense in this context because changing the static int would change it for every instance of the class (as a static int is not associated with any particular instance). Another detail I did not think about was how to implement the evasion ability of the Shadow. I eventually decided return a defense of 99 to ensure defense is always greater than any attack roll.

I initially had my takeDamage, attackOpponent and sixSidedDefence functions print out the results of each round of attacks. However, I ended up creating additional functions responsible for printing the results of each round.

Another detail from my design that was discarded in the final program was the enumerated types I had originally specified (enum Role {ATTACKER, DEFENDER};). I realized that this was unnecessary because each player's roles change from one round to the next, and updating them did not seem to affect the sequence of the game.

Implementation: see source files (assignment3.cpp, Humanoid.cpp, Goblin.cpp, Barbarian.cpp, Reptilian.cpp, BlueMen.cpp, Shadow.cpp functions.cpp driver.cpp)
Note: `./assignment3 (assignment3.cpp)` runs a game-version of the Fantasy Combat program `./driver (driver.cpp)` runs a 1000 battles of each pair of characters to test the win rate of each character. Please allow 10 seconds between each display of wins (as depicted below).

Testing: To test my program, I created a driver program (driver.cpp) that runs 1000 battles between each pair of characters. The following are win rates of the characters on the left hand column after one execution of driver.exe:

	Goblin	Barbarian	Reptilian	Blue Men	Shadow
Goblin VS	**64.9%**	**35.1%**	**0%**	**0%**	**10.1%**
Barbarian VS	**63.8%**	**49.9%**	**0%**	**0%**	**3.1%**
Reptilian VS	**100%**	**100%**	**22.6%**	**10.1**	**90%**
Blue Men VS	**100%**	**100%**	**76.1%**	**69.9%**	**100%**
Shadow VS	**100%**	**100%**	**10.1%**	**19.9%**	**40%**

Another way that I accounted for randomness:

When testing how the Goblin would cut its opponent's Achilles tendon, I commented out sections of code to remove the random number generated so that Goblin would be forced to always set the opponent's Achilles to 0.5. This serves to test that the opponent's attack was lowered by a factor of 0.5.

Similarly, to test the Shadow, I changed the probability of evasion to 100% (instead of 50%) but commenting out the `else` statement. Essentially, there would only be the `if` statement, which states that the Shadow evaded the attack. By doing this, testing should show that the Shadow never sustains any damage.

Test Case	Input	Driver Function	Expected Outcome	Observed Outcome
Valid initial command	start quit	assignment 3.cpp main()	displays character menu exiting… goodbye	displays character menu exiting… goodbye
Invalid initial commands	<ENTER> 0.99 apple	assignment3.cpp main()	Invalid. Please re-enter: Invalid. Please re-enter: Invalid. Please re-enter:	Invalid. Please re-enter: Invalid. Please re-enter: Invalid. Please re-enter:
Valid character choices	g (shown) b (shown) r m s	main()	Player 1, Goblin armor 3 strength: 8 Player 1, Barbarian armor 0 strength: 12	Player 1, Goblin armor 3 strength: 8 Player 1, Barbarian armor 0 strength: 12
Invalid character choices	a 109	main()	Invalid character. Please re-enter Invalid character. Please re-enter	Invalid character. Please re-enter Invalid character. Please re-enter
Both players choose the same character	r (reptilian) r (reptilian)	main()	Player 1, Reptilian armor 7 strength: 18 Player 2, Reptilian armor 7 strength: 18	Player 1, Reptilian armor 7 strength: 18 Player 2, Reptilian armor 7 strength: 18
Both players' characters are chosen (goblin and barbarian), and combat proceeds Run 4 times	start g <enter> b <enter <enter> start g <enter> b <enter> <enter> start g <enter> b <enter> <enter> start g <enter> b <enter> <enter>	main()	50% chance of player 1 or player 2 attacking first (random) 50% chance of player 1 or player 2 attacking first (random) 50% chance of player 1 or player 2 attacking first (random) 50% chance of player 1 or player 2 attacking first (random)	Player 1's attack! Player 2's attack Player 2's attack Player 2's attack
Game play with Blue Men vs The Shadow	m <enter> s <enter> <enter> <enter> <enter> <enter> <enter>	main()	Player 1, BlueMen armor 3 strength: 12 Player 2, Shadow armor 0 strength: 12 50% chance of player1 or 2 attacking first e.g. Blue Men attack Shadow attack should be 2-20 Shadow should either evade (50% chance) or roll defense of 1-6 Shadow either retains full health or loses 1-19 strength Shadow attacks Blue Men attack should be 2-20 Blue Men should roll defense 3-18 Blue Men either retains full health (defended) or lose 1-14 Repeats	Player 1, BlueMen armor 3 strength: 12 Player 2, Shadow armor 0 strength: 12 Player 2 attacks first e.g. Shadow strikes Blue Men with 6 points of attack force. BlueMen blocks attack with 6 defense points Damage = -3. Blue Men blocked attack, no damage. Remaining strength 12 BlueMen strikes Shadow Shadow manipulates perception and evades attack, no damage. (defense set to 99) Damage -88, Remaining strength 12 Blue Men strikes Shadow with 7 points of attack force. Shadow blocks attack with 3 defense points. Shadow was hit! 4 points of damage sustained. Remaining strength points 8 Shadow strikes Blue Men with 4 points of attack force. BlueMen blocks with 11 points of defense. Damage -1. Remaining strength 12 Blue Men strikes Shadow with 15 points of attack force. Shadow manipulates perception and evades, no damage. Remaining strength 8.

				Shadow strikes Blue Men with 8 points of attack force. Blumen blocks attack with 8 points of defense. Remaining strength 12. Blue Men strikes Shadow with 10 points of attack force. Shadow manipulates perception and evades, no damage. Damage -89. Remaining strength 8. Shadow strikes Blue Men with 4 points of attack force. BlueMen blocks with 11 points of defense. Damage -3. Remaining strength 12 Blue Men strikes Shadow with 19 points of attack force. Shadow manipulates perception and evades, no damage. Damage -89. Remaining strength 8 Shadow strikes Blue Men with 13 points of attack force. Blumen blocks attack with 18 points of defense. Remaining strength 12. Blue Men strikes Shadow with 7 points of attack force. Shadow manipulates perception and evades, no damage. Damage -92. Remaining strength 8. Shadow strikes Blue Men with 17 points of attack force. Bluemen blocks attack with 7 points of defense. Damage 7. Remaining strength 5. Blue Men strikes Shadow with 11 points of attack force. Shadow manipulates perception and evades, no damage. Damage -88. Remaining strength 8. Shadow strikes Blue Men with 9 points of attack force. Bluemen blocks attack with 12 points of defense. Damage 7. Remaining strength 5. Blue Men strikes Shadow with 15 points of attack force. Shadow blocks with 6 points of defense. Damage 9. Remaining strength -1. Player 2 Shadow slain. Player 1 BlueMen victorious!s

Reflection

Redesign: see section called Redesign and revisions above (immediately after the Design diagrams).

Difficulties Encountered: I had difficulties with determining which functions should be pure virtual. I eventually decided that the two pure virtual functions should be attackOpponent (as each Humanoid derived class attacks its opponents with different number and sides of die), and sixSidedDefense (as each derived class defends against its opponents differently). Whereas the Shadow can manipulate perception and evade attacks altogether, others defend by rolling different numbers of die.

I encountered some minor issues with pointers when I made errors during coding. For example, I created a Barbarian object Humanoid* player1pointer = new Barbarian(), then in error, entered Humanoid* player1pointer = new Goblin when the second pointer should have been player**2**pointer. (same pointer immediately assigned different memory), resulting in segmentation faults.

Lessons Learned: I learned that if a derived function accepts a pointer to a base class, that pointer cannot access protected members of the base class directly. Only this-> particular instance (can access the protected data members. I also gained a better understanding on how to decide which functions should be virtual vs which should be pure virtual and that it can be complex when member functions are called within other member functions. For instance, in my program, takeDamage is not virtual, but the sixSidedDefence function it calls is pure virtual.

References

1. Gaddis, Tony, Walters, Judy, and Muganda, Godfrey. Starting Out With C++: Early Objects. 8th Edition. Boston: Pearson, 2014. Print.
2. http://en.wikipedia.org/wiki/Goblin_(Dungeons_%26_Dragons)
3. http://en.wikipedia.org/wiki/List_of_reptilian_humanoids
4. http://wiki.belegarth.com/Nac_mac_feegle