Name: Carlos Carrillo-Calderon
Date: 12/08/2015
Course: CS_162

FINAL PROJECT

Design Description:

For this analysis, I will follow the parameters defined in the book *C++ Early Objects, 2014*. Thus, an Object-oriented analysis will be performed in order to determine the requirements for this particular system, and clarify what it must be able to do, what it needs, and how the classes created for this program are related. Also, I will specify what the classes will carry out and their responsibilities.

Description of the problem

I had to design, implement, and test a simple class hierarchy as the basis for a text-based game where the player moves through a series of rooms or compartments. The player also needs to gather items to achieve some purpose. The details are left to you!

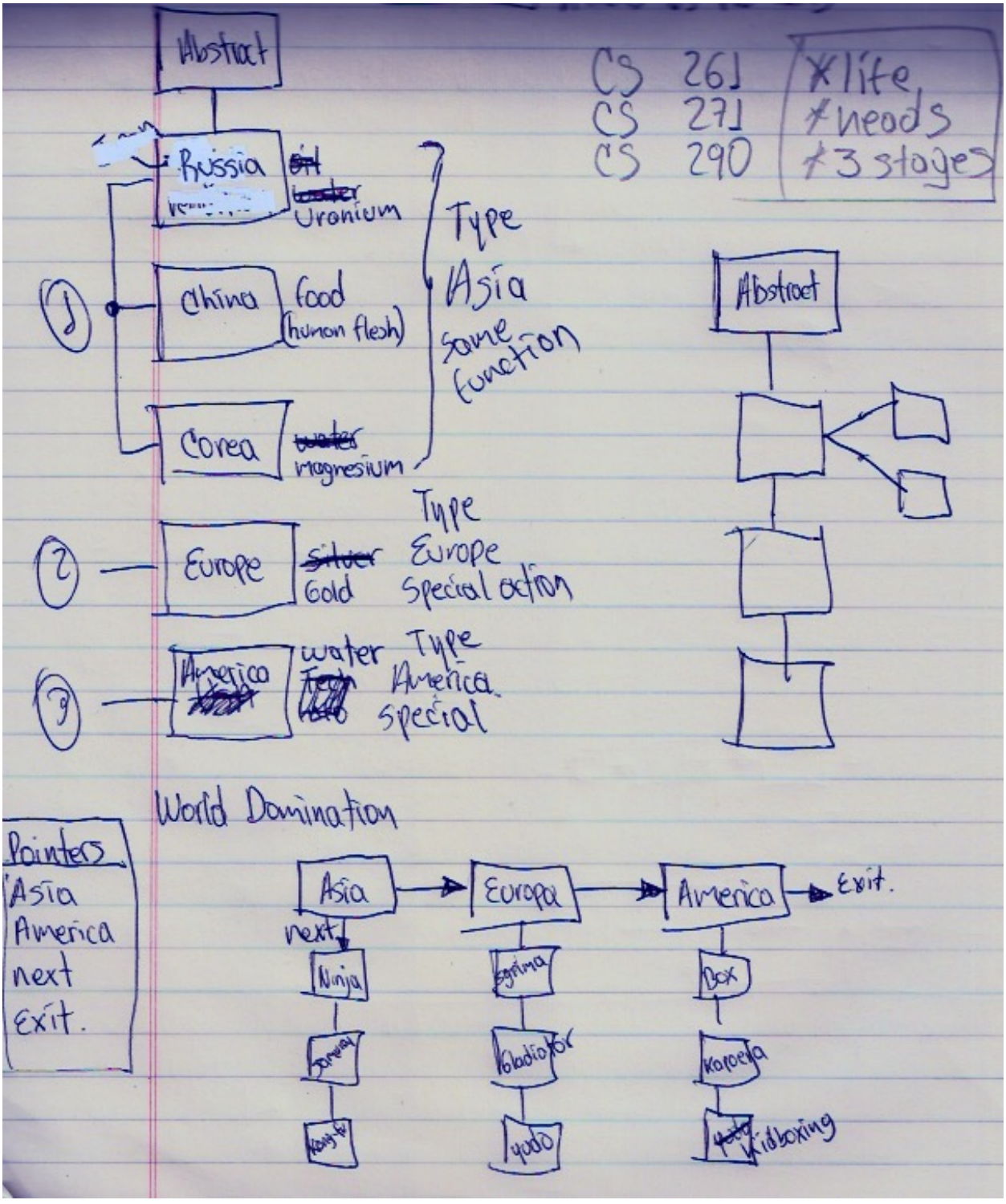
Identify classes and objects

By definition, a class, in programming, is a package that consist of data and procedures that perform operations on the data, and a set of code instruction to model real-work objects (Gaddis et al, 474). For this program, it was necessary to create a base class and 3 subclasses for each a particular virtual. The base class was an abstract class, so it was never be instantiated. Each subclass performs a special action deferent to each other. Since each subclass uses the same data elements (4 pointer variables), only one constructor was needed. As part of my design, I decided to create just a pure virtual function as well as void function whose purpose is not to use the constructor for processing all the data related to the program (I took that idea from Lab6 encryption assignment). On the other hand, I decided to include 1 more data member to manipulate data among the subclasses (the same as the pointer variables), which was associated to the player's energy points.

Initial Design (very first sketch)

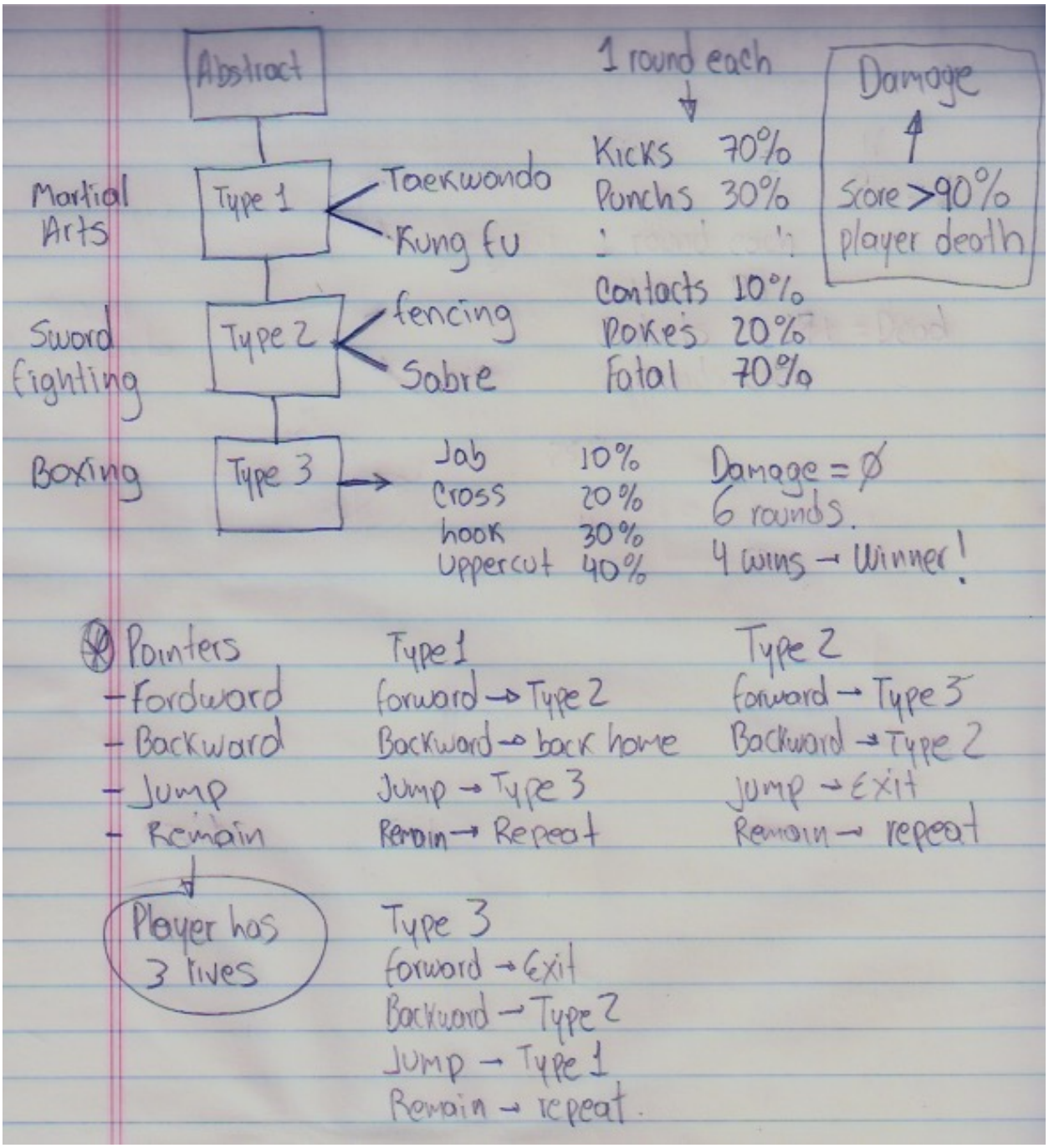
In order to learn the good habit of thinking about the application before starting to code, I crated this draft design (as requested) in which I started to build the whole concept of the game in my mind. Even though this draft is not very detailed, I may say this was the skeleton of the program. The first thing I obviously thought about was the base function. As I mentioned above, I had a clear idea about how a base class could be because of the program I made for Lab6. So my first idea (which I kept until the end of the project) was to create a virtual class to handle the different types of special actions.

Also, it was clear for me that the Player's energy variable had to definitely be a class member from the base class so all the subclasses could have access to this variable. For that reason, I would have to create a constructor for this variable anyway. Finally, I have to say that I first thought that the Armor value should also be a base class member, but later I realized I was wrong regarding that idea. Thus, this was my first sketch:



Another false assumption I had was to believe that the RNG should be part of what my fight() function had to deal with. Besides, I wouldn't have any other action to perform for my regular class function, which I called RamdomNun(). Finally, I later decided to created more class member variables for armor and stamina values but then I realized that there was no need to do so.

On the other hand, I also had to make a sketch for create a test driver program to create spaces objects, which would have to be correctly linked by means of pointers manipulation. First, since the RNG action was going to be a repetitive action, I decided to create a member function exclusively to perform this action instead of doing it every single time the user choose a particular character. As I mentioned before this function was called RamdomNun() and it was only include in the base virtual class. So this was my first sketch of the testing program:



However, as expected, I had to make several change to this initial design since certain things did not work as expected. After doing some changes (which will be mention later below) this was the final design I apply for the whole project:

Class's Attributes Definition
Now it is time to define the attributes or the data elements used to describe the objects instantiated from the classes defined above. Here are the class specifications for base class Space and the other 3 subclasses: Type1, Type2, and Type3. Additionally, I have included one class more called StackContainer, which works as the player's container for carrying items for the game.

Class name: Space
Attributes: lifeStrength //store the Energy value for the player.
*stageA //pointer variable
*stageB //pointer variable
*stageC //pointer variable
*end //pointer variable

Class name: Type1
Attributes: //the same as Space

Class name: Type2
Attributes: //the same as Space

Class name: Type3
Attributes: //the same as Space

Class name: StackContainer
Attributes: `*stackArray;` //store Items
capacity //set maximun number of elements
top //value at the top of the stack

Class's Behaviors (Functions)
Since the attributes have been defined, it is time to identify the activities or behaviors each class should perform. As already known, these behaviors are called **functions**.

1) The functions in the **Space** class should have these behaviors:

- Mutate, get, and return the double variable `lifeStrength`.
- Mutate, get, and return the four pointer variables.
- Calculate the player's life energy after interacting with one of the spaces/types.
- Generate a different random number for each subclass/type.

Consequently, these are the functions/methods to be created for the **Creature** class:

- **setLifeStrength (double):** This function mutates the member variable `lifeStrength`.

- **Double getLifeStrength():** This function returns the member variable `lifeStrength`.

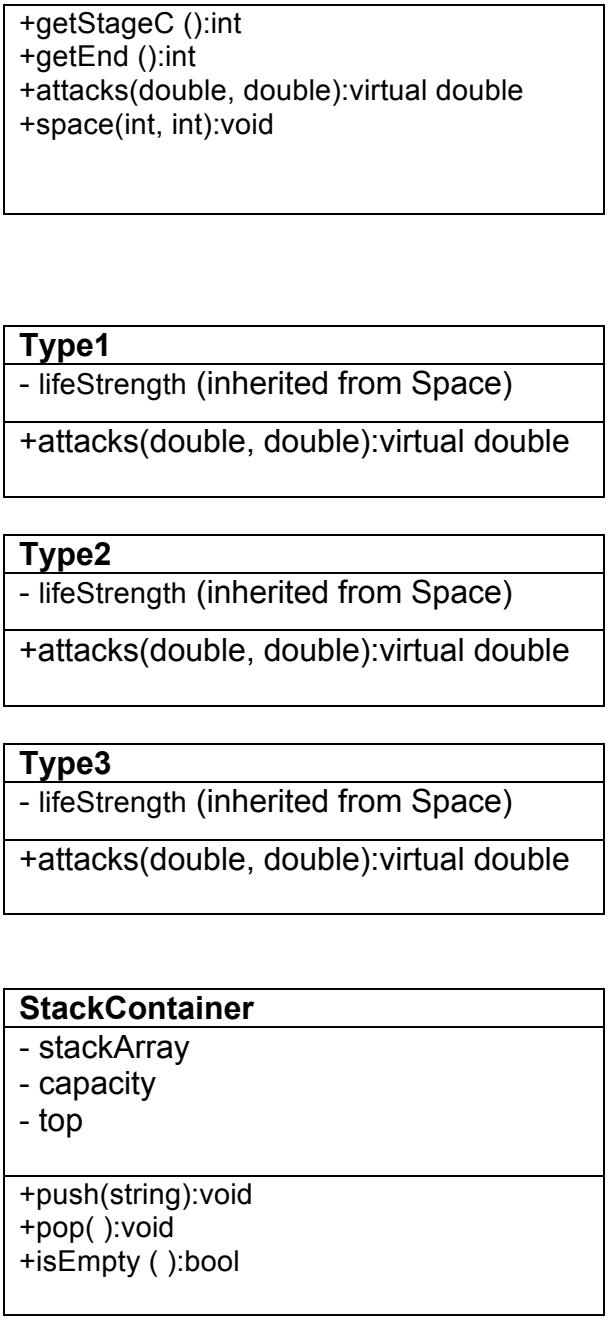
- **setStageA(int):** This function mutates the pointer member variable *stageA.
- **int getStageA ():** This function returns the pointer member variable *stageA.
- **setStageB(int):** This function mutates the pointer member variable *stageB.
- **int getStageB ():** This function returns the pointer member variable *stageB.
- **setStageC(int):** This function mutates the pointer member variable *stageC.
- **int getStageC ():** This function returns the pointer member variable *stageC.
- **setEnd(int):** This function mutates the pointer member variable *end.
- **int getEnd ():** This function returns the pointer member variable *end.
- **virtual double attacks (double, double, double, double) = 0:** This is a pure function that is redefined according to features from each combat Arena.
- **Void Space(int, int):** This function performs the actual Random number generation corresponding to each type and space.

2) The only function in the **Type1, Type2,** and **Type3** subclasses is **double attacks (double, double)** and it is obviously inherit from the base class. But, this function is polymorph according to the needs of the game.

UML Class Diagrams

Now that it has been possible to identify the class members and the behaviors that they should perform, it is also possible to draw a UML class diagram in order to visualize the **class members** and **functions** that each class should have. It is important to highlight that the minus sing to the left of each attribute indicates that it is a private member. Similarly, the plus sign to the left of each function indicates that it is a public member. Thus, this could be the UML (Unified Modeling Language) diagram for the Creature class:

Space
(protected) lifeStrength; (protected) stageA; (protected) stageB; (protected) stageC; (protected) end;
+Space(): +Space (double, int, int, int, int): +setLifeStrength (double):void +getLifeStrength ():double +setStageA (int):void +setStageB (int):void +setStageC (int):void +setEnd (int):void +getStageA ():int +getStageB ():int

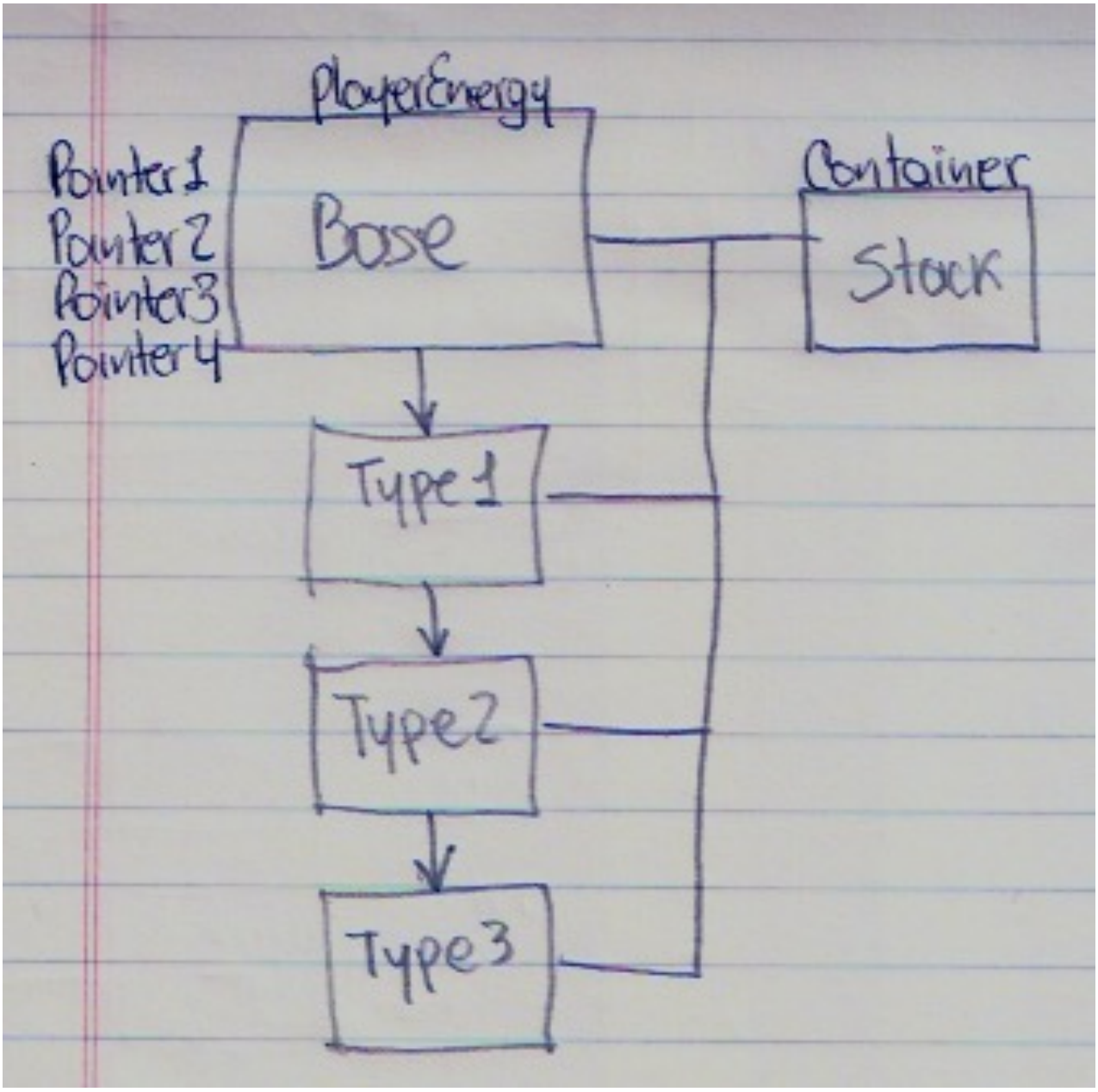


Relationships between classes

Now is time to determine the possible relationships that exist between the 6 classes. As established by Gaddis (2014), there could be three types of formal relationships among classes Access (Uses-a), Ownership (Has-a) and Inheritance (Is-a). In terms of access, the UML diagrams above clearly shows the private and public attributes that are related between the 5 classes. Thus, it could be said that there is an **Inheritance (Is-a)** relationship among the base class and the 3 children (Type1, Type2, Type3). As shown in the UML diagrams above, Space is the mother/base class of all the other 3 classes. Actually the fact the base class is an abstract class means that it can't work by itself since it cannot be instantiated. Reciprocally, the children classes can only work through

the mother-base class. Additionally, all the class members from the base class are inherited by the subclasses, so it is possible to say that this is a perfect example of an **Inheritance (Is-a)** relationship.

On the other hand, we may have an Ownership (Has-a) relationship between the abstract base class and the Stack container class. This can be established by observing the way in which the items stored in the container class are used to perform the desired actions in the different kind of subclasses.
This type of relationships can give us a clue of how a class ***hierarchy diagram*** may look like for this program:



Main Function
In order to articulate the whole design, I created this overview of the main function or driver program to create character objects, which make attack and defense rolls required to show my classes work correctly:

1. Create a main menu: In this menu I prompt the user to choose the space/room in which the player wants to start the game. In order to do this, I assigned a number to each space in the following order:

   WHAT WOULD YOU LIKE TO DO?:
   "Enter 1 to GO BACK to the SHAOLIN TEMPLE (CHINA)..
   "Enter 1 to GO BACK to the SHAOLIN TEMPLE (CHINA).."
   "Enter 3 to MOVE FORWARD to the MADISON SQUARE GARDEN (USA."
   "Enter 4 to send Predator BACK TO HIS PLANET."

2. Create the structure to link the spaces/objects: Although I may be wrong, I believed the most effective way to simulate possible linked scenarios was to use a circular do-loop. Additionally, I decided to keep the initial Energy inside the loop in order to be able to keep track of it:

```
do{
 cout <<"WELCOME TO THE PREDADOR VS. HUMANS GAME!"<<endl;

 if(choice1 == 1){ //SpaceA
      lifeStrength = 100;
     Type1 *iniObject1 = new Type1(0, 0, 0, 1, 0);
     pointer1 = iniObject1->getStageA();
     pointer2 = iniObject1->getStageB();
     pointer3 = iniObject1->getStageC();
     pointer4 = iniObject1->getEnd();
     delete iniObject3; //deallocate memory
     iniObject3 = NULL;}//clean up dangling pointer

 if(choice2 == 1){ //SpaceB
       lifeStrength = 100;
       Type2 *iniObject2 = new Type1(0, 0, 0, 1, 0);
     pointer1 = iniObject2->getStageA();
     pointer2 = iniObject2->getStageB();
     pointer3 = iniObject2->getStageC();
     pointer4 = iniObject2->getEnd();
     delete iniObject3; //deallocate memory
     iniObject3 = NULL;}//clean up dangling pointer

 if(choice3 == 1){ //SpaceC
       lifeStrength = 100;
     Type3 *iniObject3 = new Type1(0, 0, 0, 1, 0);
     pointer1 = iniObject3->getStageA();
     pointer2 = iniObject3->getStageB();
     pointer3 = iniObject3->getStageC();
     pointer4 = iniObject3->getEnd();
     delete iniObject3; //deallocate memory
     iniObject3 = NULL;}//clean up dangling pointer

 if(choice4 == 1){ //SpaceD
     pType1->setEnd(0);
```

```
        choice4 = pType1->getEnd();
        cout <<"\nEXIT = "<<choice4<<endl;}

    else
        cout <<"input validation.."<<choice2<<endl;

}while(choice1 != 0 && choice2 != 0);
```

3. Create class objects to make a single attack: For this task, I created objects in terms of the spaces in which the player is. So this was the code I used to make one of the creatures (Reptile) attacks to another creature (Barbarian):

```
//SECOND OBJECT TO SUBCLASS (TYPE 1)
predatorStrength = predatorfuerza;
Type1 *Object2 = new Type1(predatorStrength, 1, 0, 0, 0);
Object2->space(human, alien);
predatorfuerza = Object2->getlifeStrength();
delete Object2; //deallocate memory
 Object2 = NULL; //clean up dangling pointer
```

It is important to clarify that the number parameter values located next to the `predatorStrength` parameter correspond to the pointer variables used to link the spaces.

4. Create code to check container status: For this part, I used an try/catch statement to get the exception throw by the Overflow function from the Stack class:

```
try{//store trophy items in the stack/container
    cout<<endl;
    myStackContainer->push("KUNG-FU_WARRIOR'S HEAD");
    headsCounter++;}

catch(StackContainer::Overflow){//exception to detect the container is full
    if(headsCounter > 5){headsCounter = 5;}
    cout <<"PREDATOR CANNOT CUT KUNG-FU WARRIOR'S HEAD! ";
    cout <<"HIS BAG IS FULL ALREADY!!!\n"<<endl;}
```

5. Create a logic sequence for the game: Finally, I had to decided how to connect the spaces within the driver program. For that reason, I created the code according to the main argument of the story of the game. I had to use several lines of code to tell the story to the player.

THE GAME

1) The Story: after several days of thinking and creation, I finally came up with this story for the game:

"After coming to Earth in order to kill humans to collect their skulls, Predator became a public enemy for all humans. For this reason the UN has declare a resolution to challenge Predator to fight with the best skilled humans warriors from different parts of the world and different fighting disciplines. The UN has decided that these

disciplines must be TAEKWONDO, KUNG-FU, FENCING, SABRE FIGHTING, and BOXING.

Last week, NASA contacted Predator in order to propose him the challenge and he happily accepted it. His response was: “IT WOULD BE MY PLEASURE TO HAVE SOME MORE HUMAN SKULLS IN MY PERSONAL COLLECTION”. But added a difficult condition: “IF I BEAT ALL THE FIGHTERS, MY RACE WILL INVIDE YOUR PLANET AND DO WHATEVER WE WANT WITH IT”. Although it was a tough decision, the UN also accepted the challenge since all humans are very confident their warriors will defeat Predator.The combats will be performed in 3 different arenas:

* SHAOLIN TEMPLE (CHINA)
* ROMAN COLISEUM (ITALY)
* MADISON SQUARE GARDEN (USA)

Predator has to learn how to legally defeat all the challengers from each of these 3 arenas. He MUST follow the rules for each combat. Although he can choose where to start, he must also beat all the human warriors in order to claim his victory. Predator can also quit the challenge at any time! But he will have to promise never to come back; otherwise Humans will capture him forever… Enjoy the adventure!!!

2) The Rules: After being changing the rules for the majority of the designing time, I finally came up with this general rules:

GENERAL RULES:

- Only 1 player is allowed. The player controls Predator’s actions.
- Predator must have an Energy Level > 0% to finish the game.
- Predator must won at least 3 challenges in 3 different Arenas to finish the game.
- Predator must have collected at least 5 Trophy Heads to finish the game.

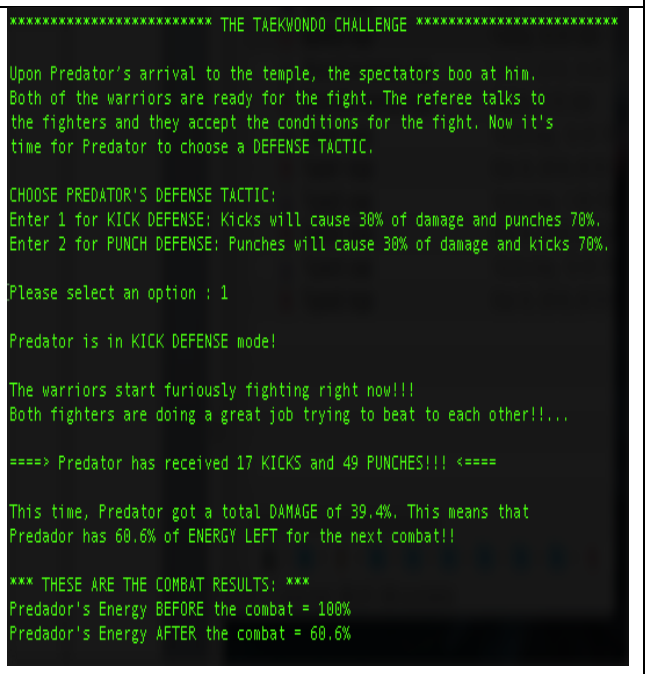
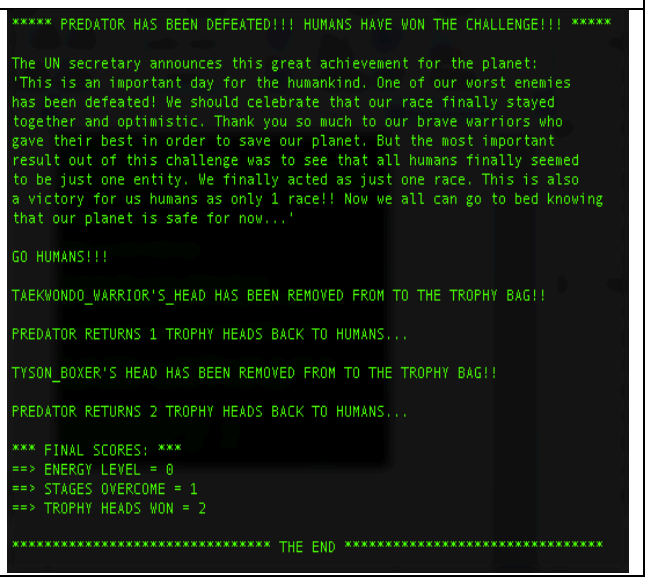
ALLOWED ACTIONS:

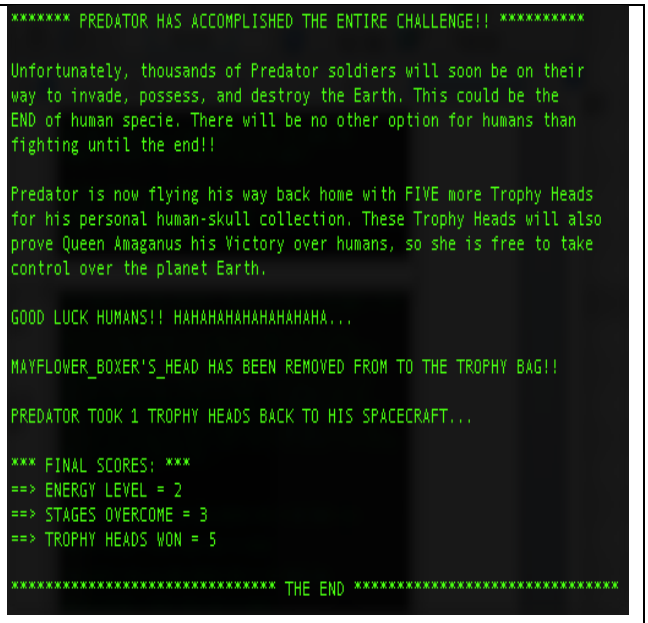
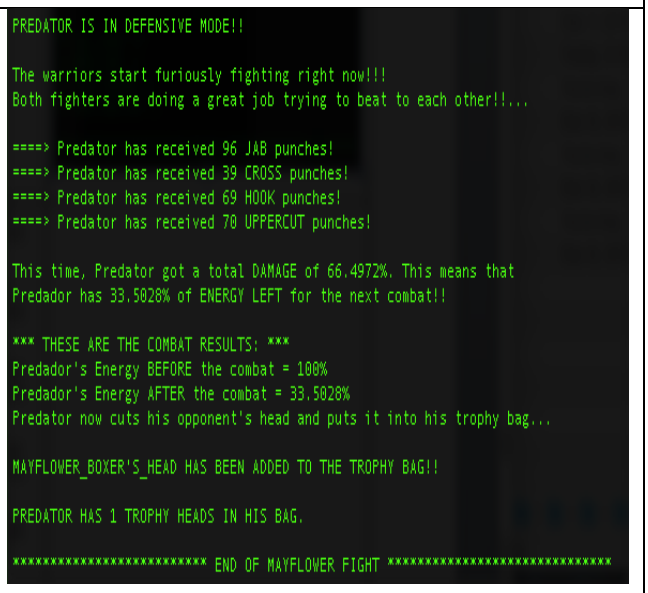
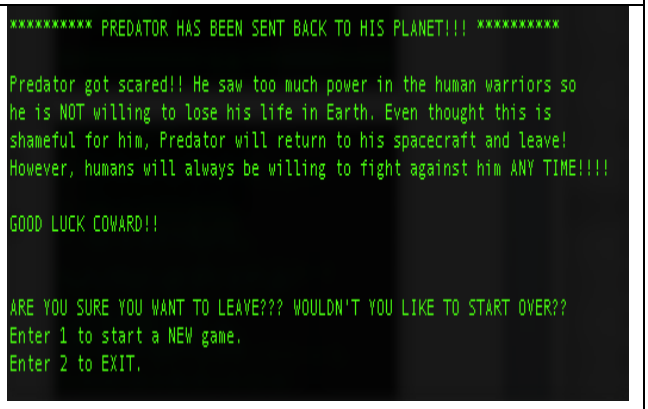
- If Predator wins a combat, he will cut his opponent’s head and keep it as a “Trophy”. He can perform this action ONLY up to 5 times.
- Predator can exchange his Trophy Heads for Armor and Energy points.
- Predator can choose fight strategies according to each particular type of challenge.
- Predator can start the game in ANY Arena the player wants.
- Predator can jump Arenas backwards and forwards.
- Predator can remain in any Arena and repeat the fights.
- Predator can leave the challenge at any time after completing an Arena.
- Predator can start over the challenge (or leave) after being defeated.

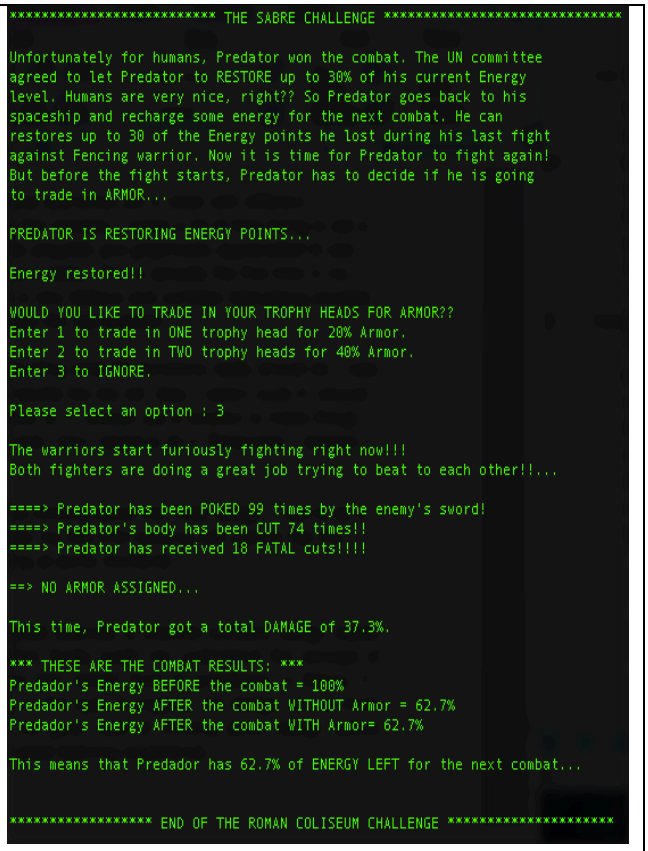
Although more rules and features are shown in the game itself, these are the more general rules the player should know in order to play the game.

Test Plan

In order to test this program, I created a 4-column table. In the first column, I described **what I was going to test**. In the second column, the possible **Input values** to perform the test. In the third column, the **expected outcomes**. And finally, in the forth column, the **actual values or output** that the program shows after running. This is the table I used. In my testing process, I basically tested all the instances of the game necessary to enter, exit or finish it. For logistic reasons, I wont include all the possible circumstances. It is also relevant to say that the results will also vary depending to the action/decision taken by the player during the game.

Section to be tested	Input provided	Expected Output	Actual Output
attacks(dob, dob, dob, dob) space(string, string)	Input to have a fight in the StageA	Correct energy points calculation for Predator. Correct defense tactic	
attacks(dob, dob, dob, dob) space(string, string)	ENERGY = 0 STAGES OVERCOME = 1 TROPHY HEADS = 2	Correct energy points calculation for Predator. Game termination due to Predator's death	

attacks(dob, dob, dob, dob) space(string, string)	ENERGY = 2 STAGES OVERCOME = 3 TROPHY HEADS = 5	Correct energy points calculation for Predator. Text announcing Predator's victory over the challenge sing all the main requirements were accomplished.	
attacks(dob, dob, dob, dob) space(string, string)	Input to have a fight in the StageC	Correct energy points calculation for Predator. Correct defense tactic. Extra energy points applied to Predator after exchange one of his trophy heads	
attacks(dob, dob, dob, dob) space(string, string)	Input to send predator back to his planet.	Correct energy points calculation for Predator. Correct text saying that predator quit the challenge so he is going back to his planet.	

attacks(dob, dob, dob, dob) space(string, string)	Input to have a fight in the StageB	Correct energy points calculation for Predator. Correct defense tactic. Extra energy points applied to Predator after exchange one of his trophy heads	

Problems while designing and implementing your program

In general, the main problem I faced was to properly set the driver program to create character objects. Since there was several scenarios (25 in total) with different parameters, I inserted a lot bugs during this process. I spent a lot of time trying to set a sort of linked structure so the program could actually be playable and graphically clear to see actual status of the player after each action. This is actually the first time in my role as a programmer that I had to face a very tedious and complex debugging process. I may say that attention to little details was the real problem.

On the other had, I struggle a little bit deciding what the functions really should do. I was not sure if the attack class should also handle the die roll. Also, I wasn't sure if I should control all the data through the constructor or if I should use a function to make things less messy. I finally decide to use both and I don't regret to do it. Personally, I believe it was a good I idea. Another issue was to figure out how to set the stack class to make it work as container for only 5 items max.

Finally, I also spent a good chunk of time thinking about the initial design. I had to do a lot of reading and research. This was also my first time creating my own game so it was had to me to try to define a clear and straightforward structure/story for the game. Although it was fun, it was also frustrating at some moments.

Conclusion:
In conclusion, this assignment was a great exercise to practice software design based on an OOD approach. Also, this program finally made me understand the concept of heritage, which has a lot of benefits but also certain level of complexity. In theory, if I follow these same steps when working on this type of projects, the design can be more accurate. Also, this reinforced on me the good habit of thinking about design before starting to code.

In general, this was a great learning experience. I believe my creativity was challenged as well as all my programing skills. I may say that I could almost use all I have learned about C++ in order to make this program work.