UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS110 - Program Design: Introduction

Assignment 2 Specifications

Release Date: 09-09-2024 at 06:00

Due Date: 11-10-2024 at 23:59

Total Marks: 400

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you start coding.***
- This assignment should be completed individually, no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- Be ready to upload your assignment well before the deadline, as **no extension will be granted.**
- If your code does not compile, you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence/absence of certain functions or classes).
- Failure of your program to successfully exit will result in a mark of 0.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at `https://portal.cs.up.ac.za/files/departmental-guide/`.
- Unless otherwise stated, the usage of c++11 or additional libraries outside of those indicated in the assignment, will **not** be allowed. Some of the appropriate files that you have submit will be overwritten during marking to ensure compliance to these requirements. **Please ensure you use c++98**
- All functions should be implemented in the corresponding `cpp` file. No inline implementation in the header file apart from the provided functions.
- The usage of ChatGPT and other AI-Related software is strictly forbidden and will be considered as plagiarism.
- Note, UML notation for variable and function signatures are used in this assignment.
- Please note that there is a late deadline which is 1 hour after the initial deadline, but you will lose 20% of your mark.

2 Overview

Polymorphism is a concept in object-oriented programming which allows programmers to define hierarchies of classes. This can be used when there are certain properties or functions which are shared between multiple classes. Polymorphism can be used to store different types of classes in the same variables by making the variable as the base type. This can be used together with abstract classes, which are classes that define a set of functions but does not give an implementation for all functions. Another class must then inherit from this class and then implement the functions that were not previously implemented.

3 Background

Dungeons & Dragons is a tabletop role-playing game where players create characters and embark on adventures in a fantasy world. For this assignment, you will be implementing a combat system which is *roughly* inspired by Dungeons & Dragons. **Note that this assignment does not follow any official rules of Dungeons & Dragons**. This assignment uses a random number generator to simulate dice and this is then used to determine combat interactions. Some of the class names and actions are *very roughly* inspired by Dungeons & Dragons, but this is all made up to show off polymorphism.

4 Tips for Assignment 2

- Since using polymorphism is such an important concept, the mark for the testing task for this assignment has been doubled. Thus 20% of this assignment's mark is for testing.
- The specification for this assignment is very large. I recommend reading through it once without trying to code it. Once you have read through it, start by implementing everything one class at a time.
- The player attack and support operators are private and return void, as chaining can be very confusing with this scenario. You can still chain the operators for the Party class.
- Parties can attack themselves, and also support parties which are not their own. You do not need to worry about this, and do not need to add any checks that would prevent this.

5 Your Task:

You are required to implement the following class diagram illustrated in Figure 1. Pay close attention to the function signatures as the `h` files will be overwritten, thus failure to comply with the UML, will result in a mark of 0. Note that you are provided with header files. **Do not change the header files.**

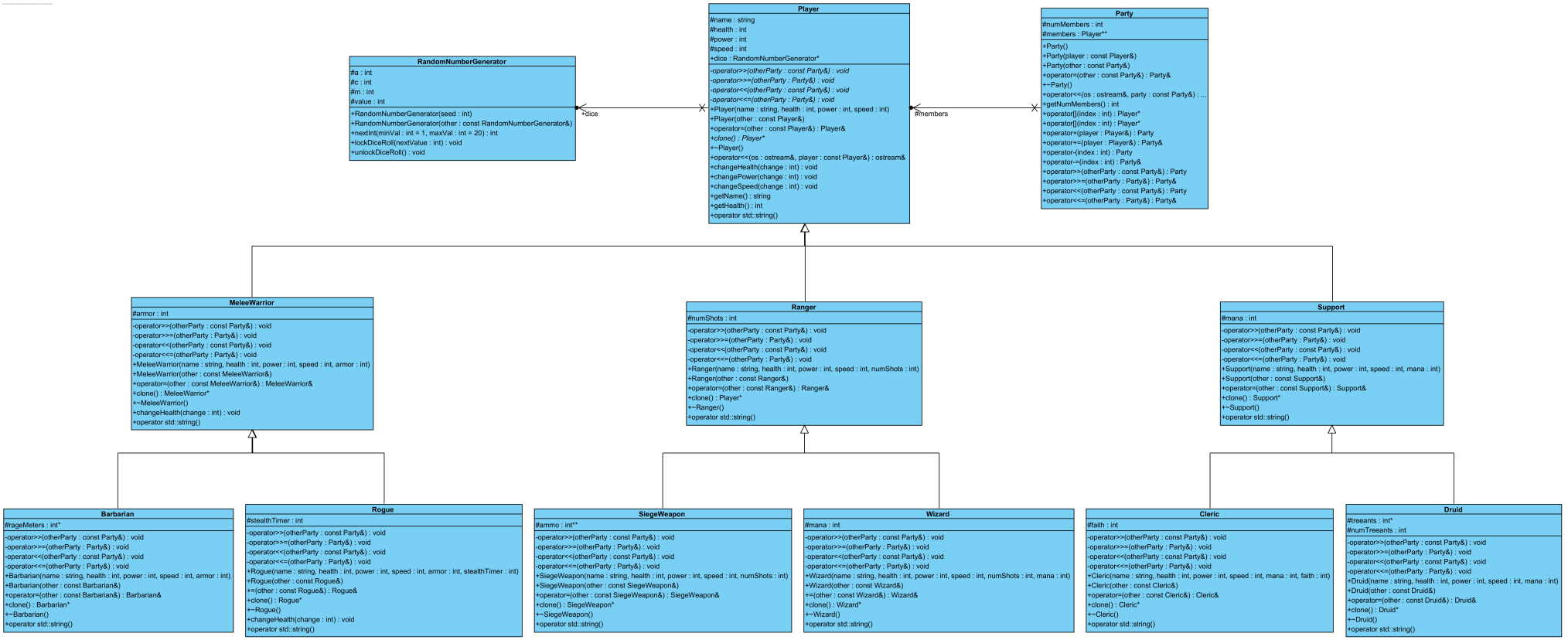

Figure 1: Class diagrams

Note, the importance of the arrows between the classes are not important for COS 110 and will be expanded on in COS 214.

5.1 RandomNumberGenerator

This is a random number generator class which is given to you. All of the functions are implemented inline and will be replaced on Fitchfork as you are not allowed to modify this class. The algorithm provided is a Linear congruential generator and uses the constants used by the `glibc` library, thus it mimics the random number generator found in `gcc`.

5.1.1 Functions

- +RandomNumberGenerator(seed: int)
  - This is the constructor for the *RandomNumberGenerator*. It takes in an int which is used to seed the random number generator.
- +RandomNumberGenerator(other: const RandomNumberGenerator&)
  - This is the copy constructor for the RandomNumberGenerator. The internal state of the random number generator is copied, which means that if the copy constructor is used, then both objects will give the same random numbers.
- +nextInt(minVal: int, maxVal: int): int
  - This generates an int value with a minimum value of *minVal*, and a maximum value of *maxVal*.
  - The *minVal* value is set to 1 by default and the *maxValue* value is set to 20 by default.
- +lockDiceRoll(nextValue: int): void
  - **This function should not be called outside of main.cpp.**
  - This function is only for testing purposes. It will lock the random number generator so that only the passed-in value will be rolled until the object is unlocked. This can be used in your testing code to force certain outcomes and will allow for easier testing.
- +unlockDiceRoll(): void
  - **This function should not be called outside of main.cpp.**
  - This function is only for testing purposes. It will unlock the random number generator so that random numbers can be generated again.

5.2 Party

This class is used to group players. Polymorphism is used to allow multiple different classes to be stored in a single array.

5.2.1 Members

- #members: Player**
  - This is a 1D dynamic array of Player pointers.
  - This array will always be perfectly sized and will **not** contain NULL.
- # numMembers: int
  - This is the size of the members array.

5.2.2 Functions

- +Party()
  - This is the default constructor. It should initialise all member variables to show that there are 0 players.
- +Party(player: const Player&)
  - This is the parameterized constructor.
  - Initialise the party such that the passed-in player is the only player in the party. A **deep copy** of the player should be made.
- +Party(other: const Party&)
  - This is the copy constructor. Deep copies of all variables should be made.
- +operator=(other: const Party&): Party&
  - This is the assignment operator. Deep copies of all variables should be made.
- ~Party()
  - This is the destructor. All dynamic memory should be freed.
- +operator«(os: std::ostream&, party: const Party&): std::ostream&
  - This is the stream-insertion operator.
  - Loop through the members array and call the stream insertion operator on every member.
  - There should be a new line between each player of the party. There should not be an endline after the last player.

- +getNumMembers() const: int
  - This is a constant function.
  - This returns the number of members (aka players).
- +operator[](index: int) const: Player*
  - This is a constant function.
  - This is the subscript operator.
  - If the index is invalid, return NULL; otherwise return the player at the passed-in index.
- +operator[](index: int): Player*
  - This is the subscript operator.
  - If the index is invalid, return NULL; otherwise return the player at the passed-in index.
- +operator+(player: Player&) const: Party
  - This is a constant function.
  - This is the addition operator. This should not change the current object.
  - This should add the passed-in player at the end of the *members* array. Note that the array needs to be resized to accommodate this.
  - The player should be added using a deep copy.
  - Return the result of adding the passed-in player to the current party.
- +operator+=(player: Player&): Party&
  - This is the addition assignment operator. This should change the current object.
  - This should add the passed-in player at the end of the *members* array. Note that the array needs to be resized to accommodate this.
  - The player should be added using a deep copy.
  - Return the result of adding the passed-in player to the current party.
- +operator-(index: int) const: Party
  - This is a constant function.
  - This is the subtraction operator. This should not change the current object.
  - This should try and remove the player at the passed-in index. If the index is invalid, then don't make any changes.
  - If the index is valid, then the array needs to be resized such that it is still perfectly sized. The memory for the removed player should be freed.
  - Return the result of removing the corresponding player from the party.

- +operator-=(index: int): Party&
  - This is the subtraction assignment operator. This should change the current object.
  - This should try and remove the player at the passed-in index. If the index is invalid, then don't make any changes.
  - If the index is valid, then the array needs to be resized such that it is still perfectly sized. The memory for the removed player should be freed.
  - Return the result of removing the corresponding player from the party.
- +operator»(otherParty: const Party&) const: Party
  - This is a constant function.
  - This is usually the right shift operator. For the context of this assignment, it is the **attack** operator.
  - This function should not change the current object or the passed-in parameter, but instead, return a copy.
  - Loop through the members and call the attack operator on each member using the passed-in parameter as the target for the attack.
  - Return *this party* after the attacks have been applied by the copy. Note that since the *Player::operator»* won't change the object its called on the returned object will just be a copy of the original.
- +operator»=(otherParty: Party&): Party&
  - This is usually the right shift assignment operator. For the context of this assignment, it is the **attack** operator.
  - This function should change the current object and the passed-in parameter.
  - Loop through the members and call the attack operator on each member using the passed-in parameter as the target for the attack.
  - Return *this party* after the attacks have been applied.
- +operator«(otherParty: const Party&) const: Party
  - This is a constant function.
  - This is usually the left shift operator. For the context of this assignment, it is the **support** operator.
  - This function should not change the current object or the passed-in parameter, but instead, return a copy.
  - Loop through the members and call the support operator on each member using the passed-in parameter as the target for the support.
  - Return *this party* after the support has been applied by the copy. Note that since the *Player::operator«* won't change the object its called on the returned object will just be a copy of the original.

- +operator«=(otherParty: Party&): Party&
  - This is usually the left shift assignment operator. For the context of this assignment, it is the **support** operator.
  - This function should change the current object and the passed-in parameter.
  - Loop through the members and call the support operator on each member using the passed-in parameter as the target for the attack.
  - Return *this party* after the support has been applied.

5.3 Player

This is an abstract class that will represent players. Polymorphism is used such that objects of the other classes can be stored in pointers of type Player.

This class has declared the Party class as a friend class.

5.3.1 Members

- # name: std::string
  - This is the name of the player.
- # health: int
  - This is the health of the player.
  - If this reaches 0 or goes negative, then the player is deemed dead.
- # power: int
  - This is the power of the player. It can also be seen as the strength of the player.
  - This value is used during attacks to determine how much damage an attack deals.
- # speed: int
  - This is the speed of the player.
  - This value is used to determine how many attack actions a player can perform per turn.
- + dice: RandomNumberGenerator*
  - This is the dice that the player will use to determine attack or support values.
  - Do not call the `nextInt()` function unless told to do so, as calling the function too few or too many times will result in wrong answers.
  - This variable is public to allow main.cpp to call `lockDiceRoll()` and `unlockDiceRoll()` on any player.

5.3.2 Functions

- +Player(name: std::string, health: int, power: int, speed: int)
  - This is the parameterized constructor. Initialise the member variables to the passed-in parameters.
  - The RandomNumberGenerator should be initialised to a seed calculated using:

```
health * power * speed + health + power + speed
```

- +Player(other: const Player&)
  - This is the copy constructor. Deep copies of all variables should be made.
- +operator=(other: const Player&): Player&
  - This is the assignment operator. Deep copies of all variables should be made.
- +clone() const: Player*
  - This is a constant function.
  - This is a pure virtual function.
  - This will be used to create deep copies of a player, this is due to the Player class being abstract and therefore cannot call the copy constructor.
- +~Player()
  - This is the destructor. All dynamic variables should be freed.
- +operator«(os: std::ostream&, player: const Player&): std::ostream&
  - This is the stream-insertion operator.
  - This should use the string casting operator to convert the player to a string. No formatting should be added to the string.
- +operator std::string() const
  - This is the string casting operator.
  - *Hint: This function was not shown in the lectures. Feel free to do some research into C++ casting operators. The signature of these functions makes it look like they have no return type, but this is because casting operators can only return the same type as the operator name. Thus, a string should be returned from this function.*
  - Return a string representation of the player. The format is as follows *(Note: there should not be any spaces or newlines. Values inside curly braces should be replaced with values and the curly braces should be removed)*:

```
{name}:health={health},power={power},speed={speed}
```

- -operator»(otherParty: const Party&) const: void
  - This is a constant function.
  - This is a pure virtual function.
  - This is the attack operator. It should not make any changes to the current object or the passed-in party.
  - This is a void function, because we do not want to allow chaining due to it causing unexpected behaviour.
- -operator»=(otherParty: Party&): void
  - This is a pure virtual function.
  - This is the attack operator. It can change both the current object and the passed-in party.
  - This is a void function, because we do not want to allow chaining due to it causing unexpected behaviour.
- -operator«(otherParty: const Party&) const: void
  - This is a pure virtual function.
  - This is the support operator. It should not make any changes to the current object or the passed-in party.
  - This is a void function, because we do not want to allow chaining due to it causing unexpected behaviour.
- -operator«=(otherParty: Party&): void
  - This is a pure virtual function.
  - This is the support operator. It can change both the current object and the passed-in party.
  - This is a void function, because we do not want to allow chaining due to it causing unexpected behaviour.
- +changeHealth(change: int): void
  - Add the passed-in value to the current health value.
- +changePower(change: int): void
  - Add the passed-in value to the current power value.
- +changeSpeed(change: int): void
  - Add the passed-in value to the current speed value.
- +getName(): std::string
  - This should return the name of the player.

- +getHealth(): int
  - This should return the health of the player.

5.4 Important Notice

Note that all the classes that follow are specialisations of the Player class. Thus, many of their function descriptions will be the same. Thus to make this document a bit shorter, some of the function descriptions will be left out. All of the functions will be explained once in the `Ranger` class, and then will be left out in the rest of the classes. The changes that each class introduces will still be mentioned in the relevant sections.

5.5 Ranger

This class is derived from the Player class. This class will be used to represent a generic ranged unit.

5.5.1 Members

- #numShots: int
  - This is the number of shots that a ranger can shoot during a single action.

5.5.2 Functions

- +Ranger(name: std::string, health: int, power: int, speed: int, numShots: int)
  - This is the paramaterized constructor. Intialise the member variables to the passed-in parameters.
- +Ranger(other: const Ranger&)
  - This is the copy constructor. Deep copies of all variables should be made.
- +operator=(other: const Ranger&): Ranger&
  - This is the assignment operator. Deep copies of all variables should be made.
- +clone() const: Player*
  - This is a constant function.
  - Return a deep copy of the current object.
- +∼Ranger()
  - This is the destructor. All dynamic variables should be freed.

- +operator std::string() const
  - This is a constant function.
  - This is the string casting operator.
  - Use the result from the parent *string casting* operator. The format of the result should be: (*Note: there should not be any spaces or newlines. Values inside curly braces should be replaced with values and curly braces should be removed*):

```
Ranger:{parent string operator},numShots={numShots}
```

- -operator»(otherParty: const Party&) const: void
  - This is a constant function.
  - This is the attack operator. It should not make any changes to the current object or the passed-in party.
  - The number of actions is calculated as the *numShots* variable multiplied by the *speed* variable. For each action, follow the attack rules described in Section 5.14.
- -operator»=(otherParty: Party&): void
  - This is the attack operator. It should make changes to the current object and the passed-in party.
  - The attack rules are the same as the `Ranger::operator»`.
- -operator«(otherParty: const Party&) const: void
  - This is a constant function.
  - This is the support operator. It should not make any changes to the current object or the passed-in party.
  - The number of actions is the *speed* variable. For each action, follow the support rules described in Section 5.15.
- -operator«=(otherParty: Party&): void
  - This is the support operator. It should make changes to the current object and the passed-in party.
  - The support rules are the same as the `Ranger::operator«`.

5.6 MeleeWarrior

This class is derived from the Player class. It is a bit stronger than a normal player and thus has a overloaded changeHealth function which takes armor into account.

5.6.1 Members

- #armor: int
  - This is the armor of the MeleeWarrior, which will let it take less damage from other players' attacks.

5.6.2 Functions

- +MeleeWarrior(name: std::string, health: int, power: int, speed: int, armor: int)
- +MeleeWarrior(other: const MeleeWarrior&)
- +operator=(other: const MeleeWarrior&): MeleeWarrior&
- +clone() const: Player*
- +~MeleeWarrior()
- +operator std::string() const
  - Format:

```
MeleeWarrior:{parent string operator},armor={armor}
```

- -operator»(otherParty: const Party&) const: void
  - The number of actions is the *speed* variable. For each action, follow the attack rules described in Section 5.14.
- -operator»=(otherParty: Party&): void
- -operator«(otherParty: const Party&) const: void
  - The number of actions is the *speed* variable. For each action, follow the support rules described in Section 5.15.
- -operator«=(otherParty: Party&): void

- +changeHealth(change: int): void
  - This function will change the health of the MeleeWarrior taking the armor into account.
  - If the passed-in parameter is positive, then add it normally since healing is not affected by armor.
  - If the passed-in parameter is negative then the armor should be taken into account. The armor value should be subtracted from the attack damage. Note that if the player has more armor than attack damage nothing happens.
  - Example:
    * If the player has 3 armor, and is attacked for 5 damage, then the player only loses 2 health.
    * If the player has 3 armor, and is attacked for 1 damage, then the player's health does not change.

5.7 Support

This class is derived from the Player class. This class mainly focuses on healing other party members, and does very little damage.

5.7.1 Members

- #mana: int
  - This is the mana of the player. This is a resource which the player uses to cast spells.

5.7.2 Functions

- +Support(name: std::string, health: int, power: int, speed: int, mana: int)
- +Support(other: const Support&)
- +operator=(other: const Support&): Support&
- +clone() const: Player*
- +~Support()
- +operator std::string() const
  - Format:

```
Support:{parent string operator},mana={mana}
```

- -operator»(otherParty: const Party&) const: void
  - The number of actions is the *speed* variable. For each action, follow the attack rules described in Section 5.14.

- -operator»=(otherParty: Party&): void
- -operator«(otherParty: const Party&) const: void
  - The number of actions is the *speed* variable. For each action, follow the support rules described in Section 5.15.
- -operator«=(otherParty: Party&): void

5.8 Wizard

This class is derived from the Ranger class. It is a specialised version of the Ranger which uses mana to cast powerful attack spells.

5.8.1 Members

- #mana: int
  - This is the mana of the player. This is a resource which the player uses to cast spells.

5.8.2 Functions

- +Wizard(name: std::string, health: int, power: int, speed: int, numShots: int, mana: int)
- +Wizard(other: const Wizard&)
- +operator=(other: const Wizard&): Wizard&
- +clone() const: Player*
- +~Wizard()
- +operator std::string() const
  - Format:

```
Wizard:{parent string operator},mana={mana}
```

- -operator»(otherParty: const Party&) const: void
  - The number of actions is the *speed* multiplied with the *numShots* variable. For each action, follow the attack rules described in Section 5.14.
- -operator»=(otherParty: Party&): void
- -operator«(otherParty: const Party&) const: void
  - The number of actions is the *speed* variable. For each action, follow the support rules described in Section 5.15.
- -operator«=(otherParty: Party&): void

5.9 SiegeWeapon

This class is derived from the Ranger class. This is a specialised version of the Ranger which does a lot of damage to multiple targets at once.

5.9.1 Members

- #ammo: int**
  - This is a dynamic 1D array of dynamic integers which will store the ammo of this player.
  - The array will always be of size *numShots*. The array can contain NULL values which will indicate an empty ammo slot.

5.9.2 Functions

- +SiegeWeapon(name: std::string, health: int, power: int, speed: int, numShots: int)
  - For the ammo member variable, create an array of size *numShots*. Initialise every entry in the array to NULL.
- +SiegeWeapon(other: const SiegeWeapon&)
- +operator=(other: const SiegeWeapon&): SiegeWeapon&
- +clone() const: Player*
- +~SiegeWeapon()
- +operator std::string() const
  - Format:

```
SiegeWeapon:{parent string operator},ammo={ammoFormatted}
```

  - ammoFormatted:
    - Start with an open square brace, and end with a closing square brace.
    - The integers should be seperated by commas, with no comma after the last value.
    - If an index is NULL, then use - for that value.
- -operator»(otherParty: const Party&) const: void
  - The number of actions is the *speed* variable. For each action, follow the attack rules described in Section 5.14.
- -operator»=(otherParty: Party&): void
- -operator«(otherParty: const Party&) const: void
  - The number of actions is the *speed* variable. For each action, follow the support rules described in Section 5.15.
- -operator«=(otherParty: Party&): void

5.10 Barbarian

This class is derived from the MeleeWarrior class. It uses rage to do more damage to other players.

5.10.1 Members

- #rageMeter: int*
  - This is a dynamic 1D array of integers which will store the rage values used by this class.
  - The array will always be of size *speed*.

5.10.2 Functions

- +Barbarian(name: std::string, health: int, power: int, speed: int, armor: int)
  - Initialise the rageMeters member variable to an array of size *speed* and fill the array with 0's.
- +Barbarian(other: const Barbarian&)
- +operator=(other: const Barbarian&): Barbarian&
- +clone() const: Player*
- +~Barbarian()
- +operator std::string() const
  - Format:

```
Barbarian:{parent string operator},rageMeters={rageMetersFormatted}
```

  - rageMetersFormatted:
    - Start with an open square brace, and end with a closing square brace.
    - The integers should be seperated by commas, with no comma after the last value.
- -operator»(otherParty: const Party&) const: void
  - The number of actions is the *speed* variable. For each action, follow the attack rules described in Section 5.14.
- -operator»=(otherParty: Party&): void
- -operator«(otherParty: const Party&) const: void
  - The number of actions is the *speed* variable. For each action, follow the support rules described in Section 5.15.
- -operator«=(otherParty: Party&): void

5.11 Rogue

This class is derived from the MeleeWarrior class. This is a player which uses stealth to buff their own attacks and defense.

5.11.1 Members

- #stealthTimer: int
  - This is an integer value that will be used to determine whether the player is in stealth mode. While in stealth mode, the player does more damage and also takes less damage.

5.11.2 Functions

- +Rogue(name: std::string, health: int, power: int, speed: int, armor: int, stealthTimer: int)
- +Rogue(other: const Rogue&)
- +operator=(other: const Rogue&): Rogue&
- +clone() const: Player*
- +~Rogue()
- +operator std::string() const
  - Format:

```
Rogue:{parent string operator},stealthTimer={stealthTimer}
```

- -operator»(otherParty: const Party&) const: void
  - The number of actions is the *speed* variable. For each action, follow the attack rules described in Section 5.14.
- -operator»=(otherParty: Party&): void
- -operator«(otherParty: const Party&) const: void
  - The number of actions is the *speed* variable. For each action, follow the support rules described in Section 5.15.
- -operator«=(otherParty: Party&): void

- +changeHealth(change: int): void
  - This function will change the health of the Rogue taking the stealth timer into account.
  - If the passed-in parameter is positive, then add it normally since healing is not affected by stealth.
  - The *stealthTimer* variable should be increased by 1. After increasing by 1, apply modulus 2**speed* to the *stealthTimer* variable.
  - If the *stealthTimer* variable is less than or equal to the *speed* variable, then the player is in stealth.
  - If the player is not in stealth, then use the MeleeWarrior `changeHealth` function with the passed in damage.
  - If the player is in stealth, then use the MeleeWarrior `changeHealth` function but with half of the damage. Note that integer division should be used.

5.12 Cleric

This class is derived from the Support class. It does a bit more damage than the normal support class and also provides more healing.

5.12.1 Members

- #faith: int
  - This is an integer value that will be used when healing other players.

5.12.2 Functions

- +Cleric(name: std::string, health: int, power: int, speed: int, mana: int, faith: int)
- +Cleric(other: const Cleric&)
- +operator=(other: const Cleric&): Cleric&
- +clone() const: Player*
- +~Cleric()
- +operator std::string() const
  - Format:

```
Cleric:{parent string operator},faith={faith}
```

- -operator»(otherParty: const Party&) const: void
  - The number of actions is the *speed* variable. For each action, follow the attack rules described in Section 5.14.

- -operator»=(otherParty: Party&): void
- -operator«(otherParty: const Party&) const: void
  - The number of actions is the *speed* variable. For each action, follow the support rules described in Section 5.15.
- -operator«=(otherParty: Party&): void

5.13 Druid

This class is derived from the Support class. The Druid does not attack or support any players directly, but rather summons treeants which are small creatures which does all of the work for the Druid.

5.13.1 Members

- #treeants: int*
  - This is an 1D array of integers representing treeants. This array will store the lifetime of each treeant.
  - The size of this array will always be *numTreeants*.
- #numTreeants: int
  - This is the current number of treeants.

5.13.2 Functions

- +Druid(name: std::string, health: int, power: int, speed: int, mana: int)
  - Initialise the treeants member variables such that there are 0 treeants.
- +Druid(other: const Druid&)
- +operator=(other: const Druid&): Druid&
- +clone() const: Player*
- +~Druid()
- +operator std::string() const
  - Format:

```
Druid:{parent string operator},treeants={treeantsFormatted}
```

  - treeantsFormatted:
    - Start with an open square brace, and end with a closing square brace.
    - The integers should be seperated by commas, with no comma after the last value.

- -operator»(otherParty: const Party&) const: void
  - If the *other party* starts with a size of zero, then return immediately.
  - Start by summoning new treeants. For every speed value, do the following:
    - Roll the dice.
    - The number of new treeants to summon is calculated as : $a = \frac{\text{diceRoll}}{8} + 1$, using integer division. The treeants will have a lifetime equal to *mana*.
    - Print out the following on its own line. *Note: Values inside curly braces should be replaced with values and curly braces should be removed.* ***Spaces*** *are shown using* _:

```
{name}_rolled_a_{diceRoll}_and_summons_{a}_treeants_with_a_lifetime_
    of_{mana}
```

    - Add the *mana* value a times to the end of the treeants array. The array should be resized such that it is perfectly sized.
    - The treeants will then attack the *other party* using the following rules:
      - Each treeant will attack once, in the order that they are in within the treeants array.
      - The damage value of the treeant is *power* multiplied with its current lifetime from the array.
      - The treeants attack their corresponding enemy. Thus the first treeant attacks the first enemy, the second treeant attacks the second enemy, etc. If the end of the enemy array is reached, then wrap around back to the first enemy.
      - When the treeants are attacking, print the following on its own line: *Note: Values inside curly braces should be replaced with values and curly braces should be removed.* ***Spaces*** *are shown using* _:

```
{name}:_treeant_{treeant index}_does_{damage}_to_{attacked
    player name}
```

      - After attacking, the lifetime value in the array is decremented by 1.
      - After all of the treeants have attacked, remove all treeants which have a lifetime of 0 from the array. Remember to resize the array, such that the array is perfectly sized.

- -operator»=(otherParty: Party&): void

- -operator«(otherParty: const Party&) const: void
  - If the *other party* starts with a size of zero, then return immediately.
  - Start by summoning new treeants. For every *speed* value do the following:
    - Roll the dice.
    - The number of new treeants to summon is calculated as : $a = \frac{\text{diceRoll}}{8} + 1$, using integer division. The treeants will have a lifetime equal to *mana*.
    - Print out the following on its own line. *Note: Values inside curly braces should be replaced with values and curly braces should be removed. Spaces are shown using _*:

```
{name}_rolled_a_{diceRoll}_and_summons_{a}_treeants_with_a_lifetime_
    of_{mana}
```

    - Add the mana value a times to the end of the treeants array. The array should be resized such that it is perfectly sized.
    - The treeants will then support the party using the following rules:
      - Each treeant will support once, in the order that they are in within the treeants array.
      - The *heal* value of the treeant is *power* multiplied with its current lifetime from the array.
      - The treeants support their corresponding player. Thus the first treeant supprts the first player, the second treeant supports the second player, etc. If the end of the *other party's* array is reached, then wrap around back to the first player.
      - When the treeants are supporting, print the following on its own line: *Note: Values inside curly braces should be replaced with values and curly braces should be removed.* ***Spaces*** *are shown using _*:

```
{name}:_treeant_{treeant index}_heals_{supported player
    name}_by_{heal value}
```

      - After supporting, the lifetime value in the array is decremented by 1.
      - After all of the treeants have supported, then remove all treeants which have a lifetime of 0 from the array. Remember to resize the array, such that the array is perfectly sized.

- -operator«=(otherParty: Party&): void

5.14 Attack actions

For each attack action, the following generic steps are followed:

1. If the *other party* starts with a size of zero, then return immediately.
2. Roll the dice.
3. Based on the dice roll value, calculate a damage number. This is different for each class and will be explained in the next subsections.
4. A player in the other party is then attacked. Note that the first attack action attacks the first member in the other party, the seccond attack action attacks the second member in the other party and so forth. When the last member has been reached, it wraps back to the first member.
5. Print out the following on its own line. *Note: Values inside curly braces should be replaced with values and curly braces should be removed. Spaces are shown using _*

```
{name}_rolled_a_{dice roll}_and_does_{damage}_damage_to_{attacked player
   name}
```

***After* all actions have been used up**, you should then check whether any attacks were lethal. If any member in the other party has a health of zero or below, remove them from the party.

Below is how the damage is calculated for each class:

5.14.1 Ranger

$$\text{damage} = \begin{cases} 3 \cdot \text{power} & \text{if diceRoll} = 20 \\ 2 \cdot \text{power} & \text{if diceRoll} \in [13, 19] \\ 1 \cdot \text{power} & \text{if diceRoll} \in [6, 12] \\ 0 & \text{if diceRoll} \in [1, 5] \end{cases}$$

5.14.2 MeleeWarrior

$$\text{damage} = \begin{cases} 3 \cdot \text{power} & \text{if diceRoll} = 20 \\ 2 \cdot \text{power} & \text{if diceRoll} \in [13, 19] \\ 1 \cdot \text{power} & \text{if diceRoll} \in [6, 12] \\ 0 & \text{if diceRoll} \in [1, 5] \end{cases}$$

5.14.3 Support

$$\text{damage} = \begin{cases} 2 \cdot \text{power} & \text{if diceRoll} = 20 \\ 1 \cdot \text{power} & \text{if diceRoll} \in [6, 19] \\ 0 & \text{if diceRoll} \in [1, 5] \end{cases}$$

5.14.4 Wizard

If the mana of the wizard is larger than 0.

- $$\text{damage} = \begin{cases} 4 \cdot \text{power} & \text{if diceRoll} = 20 \\ 3 \cdot \text{power} & \text{if diceRoll} \in [13, 19] \\ 2 \cdot \text{power} & \text{if diceRoll} \in [6, 12] \\ 1 \cdot \text{power} & \text{if diceRoll} \in [1, 5] \end{cases}$$
- After attacking the wizard's mana decreases by 1.

If the mana of the wizard is zero:

- They will **not** attack. The wizard instead gains mana equal to $\frac{\text{diceRoll}}{4} + 1$, where integer division is used. Don't print out the normal attack string. Rather print out the following:

```
{name}_rolled_a_{dice roll}_and_restored_{mana restored}_mana
```

5.14.5 SiegeWeapon

Loop through the ammo array and find the first non-NULL value. If you could not find a non-NULL value:

1. We need to refill the ammo supply. Thus don't print out the normal attack string.
2. You will need to calculate two values. The first is the number of shots to refill. This is calculated as $a = \frac{\text{diceRoll}}{4} + 1$, where integer division is used. The second is the attack value of the ammo, it is calculated as $b = \frac{21 - \text{diceRoll}}{8} + 1$, where integer division is used.
3. You should then start from the beginning of the ammo array and fill every NULL you find with b. Only fill the first a spots with b. If a is less than the number of NULLs, then there will be gaps at the end. If a is more than the number of NULLs, then all slots will be filled.
4. Print out the following:

```
{name}_rolled_a_{dice roll}_and_refilled_{number of ammo filled}_ammo
```

If you found a non-NULL value in the array, then that ammo is used. Unlike most attacks, this class will attack all players in the other party. Thus the damage calculated below, is applied to every member in the other party. After the ammo is used, delete the ammo from the array and set that ammo value to NULL.

$$\text{damage} = \begin{cases} 3 \cdot \text{power} \cdot \text{ammo value} & \text{if diceRoll} = 20 \\ 2 \cdot \text{power} \cdot \text{ammo value} & \text{if diceRoll} \in [17, 19] \\ 1 \cdot \text{power} \cdot \text{ammo value} & \text{if diceRoll} \in [6, 16] \\ 0 \cdot \text{power} \cdot \text{ammo value} & \text{if diceRoll} \in [1, 5] \end{cases}$$

Note that you should print out the normal attack string for every player that has been attacked.

5.14.6 Barbarian

The barbarian has a rageValue which is the sum of all positive values in the rageMeters array. After the damage is calculated and applied, the rageMeters are updated. Thus the rageMeters will update once per each attack action.

$$\text{damage} = \begin{cases} 4 \cdot \text{power} \cdot \text{rageValue} & \text{if diceRoll} = 20 \\ 3 \cdot \text{power} \cdot \text{rageValue} & \text{if diceRoll} \in [13, 19] \\ 2 \cdot \text{power} \cdot \text{rageValue} & \text{if diceRoll} \in [6, 12] \\ 1 \cdot \text{power} \cdot \text{rageValue} & \text{if diceRoll} \in [1, 5] \end{cases}$$

The rageMeters are updated using the following rule. For each value in the rageMeters array:

1. Roll the dice.
2. Add the following value to that index : $\frac{\text{diceRoll}}{4} - 2$, where integer division is used.

5.14.7 Rogue

At the start of each action the *stealthTimer* variable should be increased by 1. After increasing by 1, apply modulus 2**speed* to the *stealthTimer* variable. If the stealthTimer variable less than or equal to the *speed* variable, then the Rogue is in stealth. For attacking, the Rogue does double damage if they are in stealth.

$$\text{damage} = \begin{cases} 4 \cdot \text{power} & \text{if diceRoll} = 20 \\ 3 \cdot \text{power} & \text{if diceRoll} \in [13, 19] \\ 2 \cdot \text{power} & \text{if diceRoll} \in [6, 12] \\ 1 \cdot \text{power} & \text{if diceRoll} \in [1, 5] \end{cases}$$

5.14.8 Cleric

$$\text{damage} = \begin{cases} 4 \cdot \text{power} & \text{if diceRoll} = 20 \\ 3 \cdot \text{power} & \text{if diceRoll} \in [13, 19] \\ 2 \cdot \text{power} & \text{if diceRoll} \in [6, 12] \\ 1 \cdot \text{power} & \text{if diceRoll} \in [1, 5] \end{cases}$$

5.14.9 Druid

The Druid's attacks don't follow the generic rules as was explained here. Thus it was given in the Druid member functions Section.

5.15 Support actions

For *each* support action, the following generic steps are followed:

1. If the other party starts with a size of zero, then return immediately.
2. Roll the dice.
3. A player in the other party is then supported. This starts at the beginning of the players array, and then moves through it. If the end of the array is reached, then it wraps back to the start. This is explained in each subsection that follows.
4. Support actions will improve one of the three player member values. Thus they can either improve health, speed or power. Support actions have a higher chance of failing and thus have a different print regarding whether the support action worked or failed. The action words are as follows:
   - Ranger, Wizzard, SiegeWeapon
     - If successful : "_and_sped_up_"
     - If not successful : "_and_failed_to_speed_up_"
   - MeleeWarrior, Barbarian, Rogue
     - If successful : "_and_powered_up_"
     - If not successful : "_and_failed_to_power_up_"
   - Support, Cleric, Druid
     - Explained in their subsections.
5. Print out the following on its own line. *Note: Values inside curly braces should be replaced with values and curly braces should be removed. Spaces are shown using _*

```
{name}_rolled_a_{dice roll}{action words}{supported player name}
```

5.15.1 Ranger

If the Ranger rolled a 20, then it speeds up the other player by 1. If not, then it failed to speed the other player up.

5.15.2 MeleeWarrior

If the MeleeWarrior rolled a 20, then it powers up the other player by 1. If not, then it failed to power the other player up.

5.15.3 Support

- If the support has mana, then it will heal the other player using the formula:

$$\text{healing done} = \begin{cases} 3 \cdot \text{power} & \text{if diceRoll} = 20 \\ 2 \cdot \text{power} & \text{if diceRoll} \in [11, 19] \\ 1 \cdot \text{power} & \text{if diceRoll} \in [2, 10] \\ 0 & \text{if diceRoll} = 1 \end{cases}$$

  It will then print out the following on its own line:

```
{name}_rolled_a_{dice roll}_and_heals_{supported player name}_by_{healing
    done}
```

  After healing, the Support's mana decreases by 1.

- If the support does not have mana, the support instead gains mana equal to $\frac{\text{diceRoll}}{4} + 1$, where integer division is used. Print out the following on its own line:

```
{name}_rolled_a_{dice roll}_and_restored_{mana restored}_mana
```

5.15.4 Wizard

If the Wizard rolled a 13 or higher, then it speeds up the other player by 1. If not, then it failed to speed the other player up.

5.15.5 SiegeWeapon

If the Siegeweapon rolled a 13 or higher, then it speeds up the other player by 1. If not, then it failed to speed the other player up.

5.15.6 Barbarian

If the Barbarian rolled a 15 or higher, then it powers up the other player by 1. If not, then it failed to power the other player up.

5.15.7 Rogue

At the start of each action, the *stealthTimer* variable should be increased by 1. After increasing by 1, apply modulus 2**speed* to the *stealthTimer* variable. If the *stealthTimer* variable less than or equal to the *speed* variable, then the Rogue is in stealth.

- If the Rogue is in stealth and rolled a 10 or higher, then it powers up the other player by 1. If not, then it failed to power the other player up.

- If the Rogue is not in stealth and rolled a 15 or higher, then it powers up the other player by 1. If not, then it failed to power the other player up.

5.15.8 Cleric

- If the cleric has mana, then it will heal the other player using the formula:

$$\text{healing done} = \begin{cases} 4 \cdot \text{power} \cdot \text{faith} & \text{if diceRoll} = 20 \\ 3 \cdot \text{power} \cdot \text{faith} & \text{if diceRoll} \in [11, 19] \\ 2 \cdot \text{power} \cdot \text{faith} & \text{if diceRoll} \in [2, 10] \\ 1 \cdot \text{faith} & \text{if diceRoll} = 1 \end{cases}$$

  It will then print out the following on its own line:

```
{name}_rolled_a_{dice roll}_and_heals_{supported player name}_by_{healing
    done}
```

- After healing, the Cleric's mana decreases by 1.
- If the support does not have mana, the support instead gains mana equal to $\frac{\text{diceRoll}}{4} + 1$, where integer division is used. Print out the following on its own line:

```
{name}_rolled_a_{dice roll}_and_restored_{mana restored}_mana
```

5.15.9 Druid

The Druid's support doesn't follow the generic rules as was explained here. Thus it was given in the Druid member functions Section.

6 Memory Management

As memory management is a core part of COS110 and C++, each task on FitchFork will allocate approximately 10% of the marks to memory management. The following command is used:

```
valgrind --leak-check=full ./main
```

Please ensure, at all times, that your code *correctly* de-allocates *all* the memory that was allocated.

7 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, **20**% of the assignment marks will be allocated to your testing skills. To do this, you will need to submit a testing main (inside the `main.cpp` file) that will be used to test an Instructor Provided solution. You may add any helper functions to the main.cpp file to aid your testing. In order to determine the coverage of your testing the gcov [1] tool, specifically the following version *gcov (Debian 8.3.0-6) 8.3.0*, will be used. The following set of commands will be used to run gcov:

```
g++ --coverage *.cpp -o main
./main
gcov -f -m -r -j ${files}
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

We will scale this ration based on class size.

The mark you will receive for the testing coverage task is determined using Table 1:

Coverage ratio range	% of testing mark
0%-5%	0%
5%-20%	20%
20%-40%	40%
40%-60%	60%
60%-80%	80%
80%-100%	100%

Table 1: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the functions stipulated in this specification will be considered to determine your testing mark. Remember that your main will be testing the Instructor Provided code and as such, it can

[1] For more information on gcov please see `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`

only be assumed that the functions outlined in this specification are defined and implemented.

As you will be receiving marks for your testing main, we will also be doing plagiarism checks on your testing main.

8 Implementation Details

- You must implement the functions in the header files exactly as stipulated in this specification. Failure to do so will result in compilation errors on FitchFork.
- You may only use **c++98**.
- You may only utilize the specified libraries. Failure to do so will result in compilation errors on FitchFork.
- Do not include using `namespace std` in any of the files.
- You may only use the following libraries:
  - sstream
  - iostream
  - string
- You are supplied with a **trivial** main demonstrating the basic functionality of this assessment.

9 Upload Checklist

The following c++ files should be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number:

- Party.cpp
- Player.cpp
- MeleeWarrior.cpp
- Ranger.cpp
- Support.cpp
- Barbarian.cpp
- Cleric.cpp
- Druid.cpp
- Rogue.cpp
- SiegeWeapon.cpp
- Wizard.cpp
- `main.cpp`
- Any textfiles used by your `main.cpp`
- `testingFramework.h` and `testingFramework.cpp` if you used these files.

The files should be in the root directory of your zip file. In other words, when you open your zip file you should immediately see your files. They should not be inside another folder.

10 Submission

You need to submit your source files on the FitchFork website (`https://ff.cs.up.ac.za/`). All methods need to be implemented (or at least stubbed) before submission. Your code should be able to be compiled with the following command:

```
g++ *.cpp -o main
```

and run with the following command:

```
./main
```

Remember your `h` file will be overwritten, so ensure you do not alter the provided `h` files.
You have 5 submissions and your best mark will be your final mark. Upload your archive to the Assignment 2 slot on the FitchFork website. Submit your work before the deadline. **No late submissions will be accepted!**

11 Accessibility

Figure 1 shows the UML of the following classes:

- RandomNumberGenerator
- Player
- Party
- MeleeWarrior
- Ranger
- Support
- Barbarian
- Rogue
- SiegeWeapon
- Wizard
- Cleric
- Druid

The following aggregation relationships exist:

- Player uses RandomNumberGenerator.
- Party uses Player.

The following inheritance relationships exists:

- MeleeWarrior inherits from Player
- Ranger inherits from Player
- Support inherits from Player
- Barbarian inherits from MeleeWarrior
- Rogue inherits from MeleeWarrior
- SiegeWeapon inherits from Ranger
- Wizard inherits from Ranger
- Cleric inherits from Support
- Druid inherits from Support