2013

University Of Pretoria

Project Manager : Ms
M. Duncan
(mduncan@cs.up.ac.za)

[COS 110 PROJECT SPECIFICATION]

The specifications for the first years COS 110 project – a miniature adventure/RPG game.

Contents

1. Change Log 4
2. General Instructions 5
3. C++ Features 5
4. Game Overview 6
General Premise 6
Example Map – Difficulty Hard (Phase 3) 6
The Map 6
The Obstacles 6
The Creeps 6
The Sprite 7
Other 7
Setup 7
Killing Creeps And Taking Damage 8
Movement 9
General Movement Of Pieces 9
Creep Specific Movements 11
5. Minimum Classes Required 14
Game 14
Map 14
Other Information 15
Player 15
Piece 15
Other Information 15
ImmovablePiece 15
Boulder 15
Initializations And Other Information 16
Wall 16
Initializations And Other Information 16
Waypoint 16
Initializations And Other Information 16
MovablePiece 17
Other Information 17
Creep 17
Other Information 17

Critter 17
Other Information 18
Hammer 18
Other Information 18
Hunter 19
Other Information 19
Runner 19
Other Information 19
Sleeper 20
Other Information 20
Sprite 21
Other Information 21
AmmoUnit 21
Other Information 21
MeleeSprite 22
Other Information 22
RangedSprite 22
Other Information 22
Warrior 22
Other Information 22
Assassin 22
Other Information 23
Mage 23
Other Information 23
Ranger 24
Other Information 24
6. Class Hierarchy Of Minimum Required Classes 25
7. Phases 26
Phase 1 for 30% - total of 30% so far 26
Phase 2 for 30% - total of 60% so far 26
Phase 3 for 30% - total of 90% so far 26
Phase 4 for 20% - total of 110% so far 27
8. Other Information 28

1. Change Log

VERSION NUMBER	CHANGES MADE	DATE RELEASED
Version 1.0	Project Specifications created.	16 September 2013

2. General Instructions

- This project may be completed either individually or in groups of two. No groups larger than two will be allowed.
- Be ready to upload your project well before the deadlines for each phase as no extensions will be granted.
- There are four phases for the project: Phase 1 is due on 7 October, Phase 2 is due on 21 October, Phase 3 and 4 are due on 4 November.
- 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 http://www.ais.up.ac.za/plagiarism/

---

3. C++ Features

The following features should be incorporated in your project as you complete the phases, since they present good coding standards and methods:

- Constructors and destructors;
- Copy constructors and overloaded assignment operators;
- Explicitly calling base class constructors with parameters;
- Inheritance and polymorphism;
- Multiple inheritance;
- Abstract classes;
- Object aggregation;
- Friends of classes;
- Pointers;
- Arrays;
- Private and protected members;
- Exceptions;
- Operator overloading; and
- Class collaboration.

---

4. Game Overview

General Premise

You will be required to create a small adventure game in which a human player competes against a number of AI type creeps, as well as a number of obstacles such as walls. The number of creeps and obstacles are independent. The game will be played on a map of `width X height` dimensions in which the player will move their sprite and the computer will move the creeps.

The objective of the game is to move the sprite from the beginning of the map to the end of the map without dying while gaining as many points as possible.

To accomplish the objective of completing the map, the player's sprite must be moved from the starting waypoint to the end waypoint. To accomplish the objective of gaining points, the player must attack and kill as many creeps as possible while losing as little life as possible.

Example Map – Difficulty Hard (Phase 3)

The following image displays a map that has all the elements that would be available at Phase 3 of the project or also known as Hard difficulty.

The Map

```
.../.....C
././.B.../
Z/....RU.S
.///.BU..B
./........
.Z.//..E.H
./.C.....C
./..//..R.
.././/.//.
B//.......
```

The Obstacles

An obstacle is a unit on a map that prevents the human player or the computer from moving into that block, forcing a unit to go around it.

There are three types of obstacles on the map:

- The `Boulder`, represented by 'B';
- The `Wall`, represented by '/'; and
- The `Waypoint`, represented by either 'S' or 'E' for the starting or ending waypoints.

The Creeps

A creep is a computer controlled unit that acts as the enemy of the human player.

There are five types of creeps on the map:

- The `Critter`, represented by 'C';
- The `Hammer`, represented by 'H';
- The `Hunter`, represented by 'U';
- The `Runner`, represented by 'R'; and
- The `Sleeper`, represented by 'Z'.

The Sprite

A sprite is the human controlled unit that represents one of four different classes, further described in the section ***Minimum Classes Required***.

The sprite will be represented by one of the following characters:

- '`^`' to denote facing upwards;
- '`{`' to denote facing to the left;
- '`}`' to denote facing to the right; and
- '`v`' to denote facing downwards.

Controlling the sprite will take place with the following commands:

- `W/w` - rotate/move upward
- `A/a` - rotate/move left,
- `S/s` - rotate/move down
- `D/d` - rotate/move right
  - For the `Mage` class, these four commands may be followed with a number 1-3, indicating the distance to move. If no integer is specified, the distance the `Mage` must travel is implicitly 1.
- `K/k` - attack the block in front of me
- `P/p` - pass on turn. In the case of the `Assassin`, pass on the current half of turn.
- `Anything else` - ignore and wait for valid input.

Other

The only other character that should be on the map is '`.`', which denotes an empty block.

Setup

Players should be able to choose a single class as their sprite. This sprite will then be placed onto the map at the starting waypoint. From there, the player will specify whether the sprite must rotate, attack or move. When a player attacks or moves, their turn is over, meaning that rotating the sprite to face a different direction does not take up a turn and any number of rotations can be done in a single turn.

A player can also 'pass', forcing their turn to end without doing anything further. This is a valid strategic move!

Killing Creeps And Taking Damage

Some of the player classes are melee classes, meaning that a creep must be in an adjacent block in order for the sprite to attack, while other classes are ranged classes, meaning that a creep must be within a certain number of blocks in order for the sprite to attack. The range for each of the ranged sprites is described in the section ***Minimum Classes Required***.

Attacking – and moving for that matter - can only be done within the four main cardinal directions of where the sprite – or creep! – is currently standing. Clarification of this can be seen in the section on ***Movement***.

A sprite may attack a creep and any of the following scenarios may occur:

- The attack might miss, dealing zero damage;
- The creep might dodge, taking no damage;
- The creep might parry, causing the damage taken to decrease by half;
- The attack might critical hit, causing the damage dealt to double; or
- The attack might hit under normal circumstance, dealing damage equal to the power of the unit.

Attack power, hit chance and critical hit chance of the sprite classes, as well as dodge chance and parry chance of the creeps are described in the section ***Minimum Classes Required***.

Every point of damage done is added to the total score of the player for the current game.

During the turn of the creep, it can attack the player's sprite, in a similar manner to which the sprite attacks. When a creep attacks, any of the following scenarios may occur:

- The attack might miss, dealing zero damage;
- The sprite might dodge, taking no damage;
- The sprite might parry, causing the damage taken to decrease by half;
- The attack might critical hit, causing the damage dealt to double; or
- The attack might hit under normal circumstance, dealing damage equal to the power of the unit.

Attack power, hit chance and critical hit chance of the creeps, as well as dodge chance and parry chance of the sprite classes are described in the section ***Minimum Classes Required***.

Every point of damage taken is deducted from the total score of the player for the current game. It is to be noted that the score can go into the negative values!

A sprite may have a chance to regain a percentage of their maximum life every fixed number of its turns, the count starting from the first time life drops below the maximum value. When the number of turns is hit, the sprite regains an amount of life specified by their class and the counter starts again. This continues until the sprite hits 100% life, in which the counter stops and does not count up again until the sprite loses life.

The number of turns, the amount of life-regeneration and the maximum life of each sprite class is described in the section ***Minimum Classes Required***.

Life-regeneration does not count towards the total score in any way and is always rounded down to the nearest whole number.

If the sprite's life hits 0 or below 0, the sprite dies and the game ends.

Movement

General Movement Of Pieces

Each sprite class has a specified number of blocks in which it can move in each turn and can't move into a block that contains a creep or an obstacle.

A creep will only move a single block in each turn, with the exception that some creeps can move more when running away. The way in which creeps move will be described in the section ***Minimum Classes Required***.

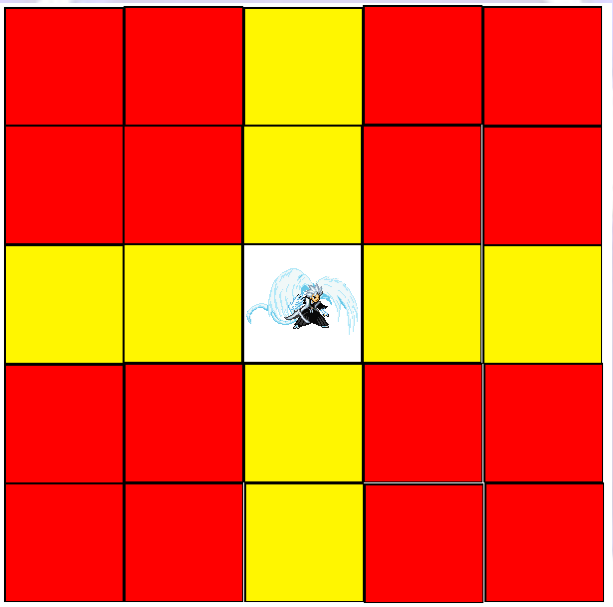
Obstacles are stationary and do not move.

When a creep dies or an obstacle – namely the `Boulder` - is destroyed, the unit should disappear from the map and the block that the unit occupied before dying becomes free, allowing other creeps or the sprite to move onto the block.

To visually clarify on the way sprites and creeps may move, consider the following image for melee units:

The unit is currently in the middle square, so it can attack or move to any of the yellow blocks. This is the case for all melee units or units with a range or move count of 1.

Similarly, consider the following image for ranged units:



Again, the unit is in the middle square and can attack or move to any of the yellow blocks. The image above displays for a range or move count of 2, but the same concept can be applied to any range.

Creep Specific Movements

Random Movement Around Initial Position

Some creeps will move randomly around their initial position but might run into a problem in which they want to move to a block that is occupied by another piece or their move takes them too far from their initial position. When this happens, the following pseudocode must be followed to move the creep.

```
get a random integer num from the given random() function

//random() returns a value from 0 to 100 inclusive
if num is between 1 and 25 inclusive
   propose to move up
else if num is between 26 and 50 inclusive
   propose to move down
else if num is between 51 and 75 inclusive
   propose to move left
else if num is between 76 and 100 inclusive
   propose to move right
else if num is 0
   do not move

ensure that both the proposed height and width values are within a range of
1 of the initial position

//The following ensures that the proposed position is available to move to
if proposed height is higher than my current height position
   if the block above me is empty
      keep proposed height position
   else if the block above me is occupied
      if the block below me is empty
         change proposed height position to the block below me
      else if the block below me is occupied
         if the block to my left is empty
            change proposed width position to the block to my left
         else
            change proposed width position to the block to my right
else if proposed height position is lower than my current height position
   if the block below me is empty
      keep proposed height position
   else if the block below me is occupied
      if the block to my left is empty
         change proposed width position to the block to my left
      else if the block to my left is occupied
         if the block to my right is empty
            change proposed width position to the block to my right
         else
            change proposed height position to the block above me
else if proposed width position is to the left of my current width position
   if the block to my left is empty
      keep proposed width position
   else if the block to my left is occupied
      if the block to my right is empty
         change proposed width position to the block to my right
      else if the block to my right is occupied
         if the block above me is empty
            change proposed height position to the block above me
         else
            change proposed height position to the block below me
```

```
else if proposed width position is to the right of my current width
position
   if the block to my right is empty
      keep proposed width position
   else if the block to my right is occupied
      if the block above me is empty
         change proposed height position to the block above me
      else if the block above me is occupied
         if the block below me is empty
            change proposed height position to the block below me
         else
            change proposed width position to the block to my left
```

Runner - Moving Back To Initial Position

There is a creep that will run away from the sprite and it is known as the `Runner`. More details can be found under its class description in the section ***Minimum Classes Required***.

When a `Runner` is too far from the initial position due to running away, it needs to move back into range if it has not been attacked again. This movement is dictated by the following pseudocode.

```
if I have not been attacked in sprite's last turn
   if I am higher than my initial position by more than 1 block
      change my current height position to the block below me
   else if I am lower than my initial position by more than 1 block
      change my current height position to the block above me
   else if I am further left than my initial position by more than 1 block
      change my current width position to the block to my right
   else if I am further right than my initial position by more than 1 block
      change my current width position to the block to my left
```

Sprite Hunting

When a creep needs to hunt the sprite, there are times when it will not be able to move to its first choice because some other piece may be in its place. When this occurs, the following pseudocode must be followed to move the creep.

```
if the sprite is above me
   move up BUT if something is above me
      if the sprite is to my left or shares the same width value
         move left BUT if something is to my left
            move right BUT if something is to my right
               move down
      else if the sprite is to my right
         move right BUT if something is to my right
            move left BUT if something is to my left
               move down
else if the sprite is below me
   move down BUT if something is below me
      if the sprite is to my left or shares the same width value
         move left BUT if something is to my left
            move right BUT if something is to my right
               move up
      else if the sprite is to my right
         move right BUT if something is to my right
            move left BUT if something is to my left
               move up
else if the sprite is to my left
   move left BUT if something is to my left
```

```
      if the sprite is above me or shares the same height value
         move up BUT if something is above me
            move down BUT if something is below me
               move right
      else if the sprite is below me
         move down BUT if something is below me
            move up BUT if something is above me
               move right
else if the sprite is to my right
   move right BUT if something is to my right
      if the sprite is above me or shares the same height value
         move up BUT if something is above me
            move down BUT if something is below me
               move left
      else if the sprite is below me
         move down BUT if something is below me
            move up BUT if something is above me
               move left
```

It is to be noted that this code alone, given the right circumstances, will cause a creep to move from one block to another and back continuously. An example can be seen in the following three game frames.

```
 -----
|v..BU|
|.....|
|.....|
|.....|
|....E|
 -----
```

```
 -----
|v..B.|
|....U|
|.....|
|.....|
|....E|
 -----
```

```
 -----
|v..BU|
|.....|
|.....|
|.....|
|....E|
 -----
```

This should not happen and needs to be taken into account with a history of the creep's movements.

The history should store up to 3 past positions.

Even with the history, it can still happen that a creep can get stuck, i.e. it needs to move to a past position in order to move at all. In the event that this does happen, the creep must remain still for the current turn but can forget the oldest position that it has been to.

The code given above may also end up proposing a position that is already occupied by another `Piece`. Recursion or iteration will need to be used in order to find a new position with the given code. This also means that you will need some method of recording what positions have already been considered and rejected, such that a new position can be found. When all positions have been considered, the creep is considered to be stuck and the above statement holds true as well.

---

5. Minimum Classes Required

Game

The `Game` class will be the main interaction point between the `Map`, the player and their sprite and the creeps. It will be the class that takes in input from the user, it will display the map for the user, it will notify the player when the game has ended and with what score and any other game controlling mechanisms that you might require.

A `Game` will have a single map and a single player with their sprite and current score.

Map

The `Map` class is the actual level that the player's sprite will move around within.

This class will have a `width X height` 2D array that acts as the map. This array will hold the positions of the waypoints, the creeps, the obstacles and the player's sprite as the game progresses. It is recommended to use `Piece***` as the array so that you have a 2D array of pointers to the pieces of the map. You could use references instead of pointers, in which case you could use a `Piece**`.

This class will also have to store the creeps and obstacles, since the creeps and obstacles are specific to a single map.

A `Map` must also be restartable, such that a player can restart and retry a map. The `Map` will restart in the event that the player steps back onto the start waypoint after having moved off of it. Everything on the map must be reset to its initial values as stated in the map files - also known as their initial positions and values - with the exception of the random function, which should NOT be reset to the beginning of its pseudo-random list. This random function is given to you and more information can be found in the section ***Other Information***.

Other Information

A set of maps has been provided, which need to be read in. It will be up to you how you read in the map files.

The maps are in the range of Easy, Medium and Hard and correspond to the different phases of the project.

Player

The `Player` class represents the human player and will hold a reference or a pointer to the sprite that they control, as well as the total score for the current game.

Piece

The `Piece` class is any element that belongs on a map. A `Piece` can represent a creep, an obstacle, a waypoint and the player sprite.

Other Information

All pieces will have a maximum and current life force that is specified in each individual class below, which will be counted as whole numbers only.

All pieces will have a type that defines what they are. This type is an enumeration that is given to you in *PieceType.h* and more information can be found in the section ***Other Information***.

Each piece will have a single character that will be displayed on the board to represent them, with the exception of the player's sprite. The sprite can be represented by 1 of four characters, as given in the section ***Game Overview – Example Map***. The characters for each other type is also provided in this section.

All pieces will have a number of moves that it can complete in a turn, which may range from 0 to infinity but will most likely have a bounded number less than 5. The three moves that will end a player's turn is moving from one block to another, attacking and passing on their turn. The moves that will not end a player's turn are the rotations of the sprite to face a new direction.

ImmovablePiece

An `ImmovablePiece` is a `Piece`.

It is an element that does not move and is used to represent the obstacles on the level.

Boulder

A `Boulder` is an `ImmovablePiece`.

A `Boulder` is an obstacle that can't be bypassed except by a `Mage`.

Initializations And Other Information

Maximum Life: 5

Attacking a boulder costs the player in score points the total amount of attack power that their sprite has but killing a boulder with a critical hit will award 150 score points.

Wall

A `Wall` is an `ImmovablePiece`.

A `Wall` is an obstacle that can't be bypassed except by a `Mage`.

Initializations And Other Information

Maximum Life: `INT_MAX`

Should a sprite actually attack a wall, it takes no damage but the sprite will lose 10% health in the event that the sprite's attack does not miss. This amount is rounded down to the nearest whole number. This loss of life is still deducted from the player's total score.

In addition to the above, regardless if the sprite's attack hits or not, the sprite will be knocked backwards by two blocks with the following restrictions:

```
If there is an obstacle at the sprite's range 1
   The sprite will take 10 damage, the player will lose 10 score points and
       the sprite will remain where it is
If there is an obstacle at the sprite's range 2
   The sprite will take 5 damage, the player will lose 5 score points and
       the sprite will move backwards to its range 1 while facing the same
       direction as before the move
If there is no obstacle at the sprite's range 1 or 2
   The sprite will take no damage and will move backwards to its range 2
       while facing the same direction as before the move
If there is a creep at the sprite's range 1
   The sprite will take 10 damage, the player will lose 10 score points, the
       creep will take 20 damage and the sprite will remain where it is
If there is a creep at the sprite's range 2
   The sprite will take 5 damage, the player will lose 5 score points, the
       creep will take 10 damage and the sprite will move backwards to its
       range 1 while facing the same direction as before the move
If there is no creep at the sprite's range 1 or 2
   The sprite will lose no life and will move backwards to its range 2 while
       facing the same direction as before the move
```

Waypoint

A `Waypoint` is an `ImmovablePiece`.

A `Waypoint` is either the starting point of the map or the ending point of the map.

Initializations And Other Information

Maximum Life: 1

Creeps inherently fear waypoints and will never step onto one, no matter the situation that they are in.

If a waypoint is attacked by a sprite for any reason and that attack hits, the waypoint will be destroyed, the player will lose 500 points from their score and they will automatically lose the game.

When the sprite is spawned, it will be facing downwards. If a sprite steps back onto the starting waypoint after having moved off of it, the map should restart, respawning all creeps, obstacles and sprite with full health and in initial positions.

When the sprite moves onto the ending waypoint, it will display an arrow facing the direction that the sprite was going.

MovablePiece

A `MovablePiece` is a `Piece`.

It is an element that is either the player's sprite or is a creep.

Other Information

All movable pieces will have a range in which they can attack, an attack power, a chance to hit, a chance to critical hit, a chance to dodge and a chance to parry.

When a movable piece is attacked, its stats are evaluated in the following order:

```
Can I dodge the incoming attack?
   YES : I take no damage!
   NO : Can I parry the incoming attack?
      YES : I take half damage.
      NO : I take full damage dealt...
```

When a movable piece is attacking, its stats are evaluated in the following order:

```
Can I hit on this attack?
   YES : Can I critical hit on this attack?
      YES : I do double damage!
      NO : I do normal full damage.
   NO : I do no damage...
```

Creep

A `Creep` is a `MovablePiece`.

A `Creep` is the AI of the computer that is the enemy of the player's sprite.

Other Information

All creeps will have an initial position that they know of as some of them will always move around this position.

Critter

A `Critter` is a `Creep`.

A `Critter` is an AI unit that is the simplest enemy unit.

Other Information

Maximum Life: 5

Move Count: 0

Range: 1

Attack Power: 10

Hit Chance: 100%

Critical Hit Chance: 50%

Dodge Chance: 10%

Parry Chance: 20%

The critter creep is the simplest creep that can oppose the player. A critter does not move at all and only attacks a sprite in range if it was attacked itself.

Hammer

A `Hammer` is a `Creep`.

A `Hammer` is an AI unit that will randomly move around its initial position within a range of 1 unless it is threatened.

Other Information

Maximum Life: 100

Move Count: 1

Range: 2

Attack Power: 40

Hit Chance: 50%

Critical Hit Chance: 5%

Dodge Chance: 2,5%

Parry Chance: 5%

The hammer creep is large, slow and unwieldy and will randomly move around its initial position within a range of 1 until it is attacked, in which case it will hunt down the sprite in an attempt to kill it.

When a hammer creep is successfully killed, the player will be awarded an additional 150 score points.

Hunter

A `Hunter` is a `Creep`.

A `Hunter` is an AI unit that actively chases down the sprite in order to kill it.

Other Information

Maximum Life: 25

Move Count: 1

Range: 1

Attack Power: 10

Hit Chance: 75%

Critical Hit Chance: 10%

Dodge Chance: 10%

Parry Chance: 20%

A hunter creep will become stealthed – visually disappearing from the map - when a sprite comes within a range of 2 of it and will remain in its position while stealthed. It is not immune to damage while stealthed and can take damage if the sprite manages to attack it. When a hunter creep takes damage while in stealth, it will become visible again and takes double damage AFTER hit chance and critical chance has been calculated from the sprite.

A hunter creep can only stealth if it went undamaged in the player's last turn and it will never choose to stealth if it can attack instead.

While a hunter creep is stealthed, it only has two options:

- Either it must attack a player – doing double damage AFTER hit chance and critical chance has been calculated - and lose its stealth; or
- It must pass on its turn and remain stealthed.

These two conditions may cause the hunter creep to remain stealthed forever if the players loses track of the creep or decides to bypass it.

Runner

A `Runner` is a `Creep`.

A `Runner` is an AI unit that will randomly move around its initial position within a range of 1 unless it is threatened.

Other Information

Maximum Life: 30

Move Count: 1

Range: 1

Attack Power: 10

Hit Chance: 85%

Critical Hit Chance: 7,5%

Dodge Chance: 5%

Parry Chance: 10%

The runner creep is a passive and defensive creep, only attacking when it is threatened. It will continue to randomly move around its initial position within a range of 1.

If the runner creep was attacked in the player's last turn, the runner creep will attack:

- If damage is dealt, the runner creep will resume its random movement on its next turn until it is attacked again.
- If no damage is dealt after attacking, the runner creep will run away from the sprite by two blocks in panic, dealing 5 damage to itself but doubles its critical hit chance for its next attack. The critical hit chance will return to normal after the next attack regardless of whether that attack deals damage or not.

In the event that the runner creep ends up being further than range 1 of its initial position, it will first attempt to return to its initial position before resuming its random movement but attacking/running away will take precedence if the sprite attacks again.

When a Runner attempts to flee, if there is an object at its range 2, it will move away from the sprite by 1 block and if there is an object at its range 1, it will not move. This does not affect the damage it inflicts upon itself.

Sleeper

A `Sleeper` is a `Creep`.

A `Sleeper` is an AI unit that will sleep – or pass its turn – until it is attacked by a player, and then actively hunts down the sprite in order to kill it.

Other Information

Maximum Life: 40

Move Count: 1

Range: 1

Attack Power: 5

Hit Chance: 75%

Critical Hit Chance: 10%

Dodge Chance: 10%

Parry Chance: 20%

When a sleeper creep is attacked while it is still sleeping, it has no chance to dodge or parry and takes the full damage dealt by the sprite.

After each time the sleeper creep takes damage, including the time it wakes up:

- Its attack power doubles; and
- Its hit chance, critical hit chance, dodge chance and parry chance all increase by 5%.

The percentages can't increase more than 100% but the attack power has no upper bound!

After each time the sleeper attacks, regardless of whether damage is dealt or not:

- Its attack power halves; and
- Its hit chance, critical hit chance, dodge chance and parry chance all decrease by 5%.

All of the sleeper's stats can never drop below their original values.

Sprite

A `Sprite` is a `MovablePiece`.

A `Sprite` represents the piece that the human player moves around the map.

Other Information

All sprites have a life regeneration percentage and a life regeneration turn count.

AmmoUnit

An `AmmoUnit` represents a unit with ammunition. Only some of the weaker sprites are `AmmoUnits` in addition to being a `MeleeSprite` or a `RangedSprite`.

Other Information

All ammunition sprites have a maximum and current ammunition amount, as well the amount by which they increase a unit's attack power.

All ammunition will always decrease by 1 every turn and ammunition will always be used during an attack if there is any ammunition available. Whether an attack hits or not has no effect on the decrease of ammunition.

Ammunition can't be regained the way life can with life regeneration.

MeleeSprite

A `MeleeSprite` is a `Sprite`.

A `MeleeSprite` is a melee unit class for a player's sprite.

Other Information

Range: 1

RangedSprite

A `RangedSprite` is a `Sprite`.

A `RangedSprite` is a ranged unit class for a player's sprite.

Other Information

Range: 2

Warrior

A `Warrior` is a `MeleeSprite`.

A `Warrior` is the most basic unit that a player can use as a sprite. A Warrior has high life and average defence with high hit and parry chances.

Other Information

Maximum Life: 250

Move Count: 1

Attack Power: 20

Hit Chance: 95%

Critical Hit Chance: 5%

Dodge Chance: 2,5%

Parry Chance: 25%

Life Regeneration Turn Count: 10

Life Regeneration Amount: 10%

Assassin

An `Assassin` is a `MeleeSprite` and an `AmmoUnit`.

An `Assassin` is a wily melee unit that attacks with poisoned blades for extra damage.

Other Information

Maximum Life: 150

Move Count: 2

Attack Power: 5

Hit Chance: 95%

Critical Hit Chance: 20%

Dodge Chance: 50%

Parry Chance: 20%

Life Regeneration Turn Count: 5

Life Regeneration Amount: 10%

Maximum Ammunition Amount: 25

Attack Power Increase: 20

The assassin class can move or attack twice per turn, in any combination. The assassin class can also obviously pass on one or both of its 'turns' per one of the player's turn.

Mage

A `Mage` is a `RangedSprite`.

A `Mage` is a ranged unit that has very poor defence and low life but has a high attack rating.

Other Information

Maximum Life: 100

Move Count: 1

Attack Power: 30

Hit Chance: 80%

Critical Hit Chance: 35%

Dodge Chance: 5%

Parry Chance: 0%

Life Regeneration Turn Count: 7

Life Regeneration Amount: 15%

If a mage sprite wants to move in a turn, it may teleport to any block within a range of 3 as long as the destination block is unoccupied by either obstacle or creep. This teleport can bypass or 'jump over' all walls, boulders and creeps.

Ranger

A `Ranger` is a `RangedSprite` and an `AmmoUnit`.

A `Ranger` is a ranged unit that attacks from distance with its bow and arrow.

Other Information

Maximum Life: 200

Move Count: 1

Attack Power: 15

Hit Chance: 75%

Critical Hit Chance: 15%

Dodge Chance: 15%

Parry Chance: 5%

Life Regeneration Turn Count: 3

Life Regeneration Amount: 7%

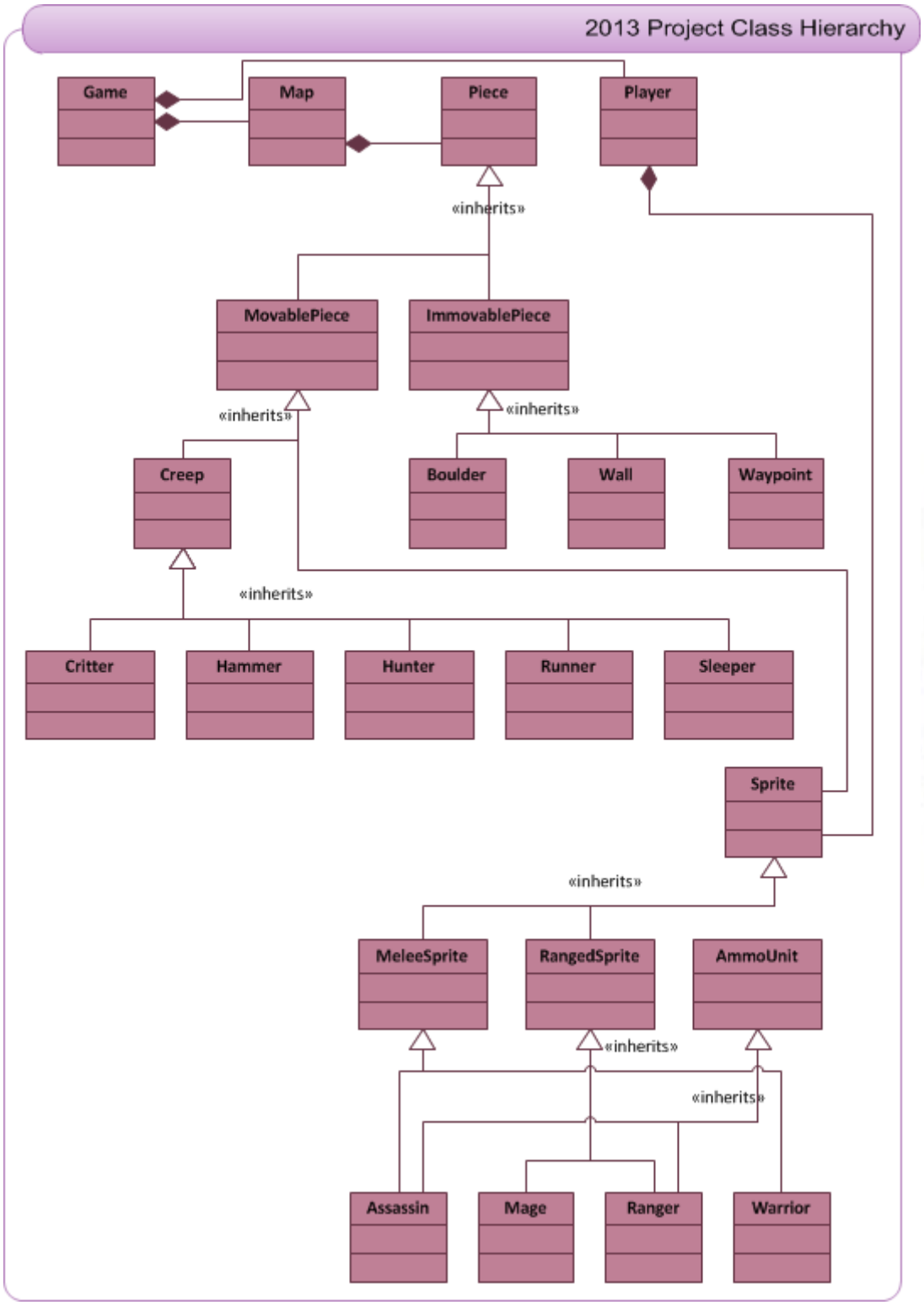
Maximum Ammunition Amount: 30

Attack Power Increase: 10

When a ranger sprite comes under attack, regardless of whether it takes damage or not, its critical hit chance will fluctuate. The critical hit chance will double for the next turn, halve from its initial value for the following turn and finally return to normal on the 3rd turn following the attack.

A diagram of the above information has been given in the next section for a visual representation of the minimum classes that are required and their relationships.

6. Class Hierarchy Of Minimum Required Classes



7. Phases

Phase 1 for 30% - total of 30% so far

Classes to implement:

- `Game`
- `Map`
- `Player`
- `Piece`
- `ImmovablePiece`
- `Waypoint`
- `MovablePiece`
- `Sprite`
- `MeleeSprite`
- `Warrior`

These are the classes that will be required to be implemented.

Seeing as there are no attackable objects - with the exception of the waypoints – attacking, defending and life regeneration for the sprites and defending for the obstacles is not required for this phase. Only movement will be required for the sprites.

Phase 2 for 30% - total of 60% so far

Classes to implement:

- `Boulder`
- `Wall`
- `AmmoUnit`
- `RangedSprite`
- `Assassin`
- `Mage`
- `Ranger`
- `Creep`
- `Critter`

Pieces should now be able to attack and defend in this phase. Life regeneration should also be implemented at the end of this phase.

Phase 3 for 30% - total of 90% so far

Classes to implement:

- `Hammer`
- `Hunter`
- `Runner`
- `Sleeper`

Phase 4 for 20% - total of 110% so far

This phase will be for impressive features in the game and will be marked manually. Some features to implement may include:

- Map generation at the same level as the Hard maps, thus including all the different types of obstacles and creeps.
- Implementing extra skills for each of the sprites.
- Implementing a graphical user interface for the game in which the game still conforms to all specifications.

If there is a feature that you believe might be impressive enough, feel free to email your idea to assistant lecturer Megan Duncan and she will inform you if it will be enough to grant you the marks – if implemented correctly – or not.

8. Other Information

- Phase 3 and Phase 4 upload slots will be opened at the same time. The phase 4 fitchfork hurdle must be passed before manual marking will occur.
- You will be required to design and implement the minimum required classes, as well as any other classes that you believe you might need to complete the project.
- You will be required to create your own `makefile` for this project. The only restriction is that compilation and linking MUST be done with the `static` flag and the executable that is generated MUST be named *RPG* at all times, throughout all phases.
- The code for printing out the map and for generating the chance percentages have both been provided. The code for printing the map should go in the `Map` class and the code for the random number generation should go in the `Piece` class but if you place it anywhere else and it still works, then that is your choice. These two functions are NOT allowed to be altered in any way with the only exception being the scope resolution operator. If the functionality or output code is changed in any way, you WILL lose marks with fitchfork.
- The randomness throughout the project depends on the seeds provided to each class, thus the enumeration on the classes has been provided to you as the header file *PieceType.h*. If you need to add more enumerations, ensure that they are added to the END of the enumeration list so as not to affect the inner workings of the project. With regards to using the random function provided, the order in which you call the function must conform to the following:
    - Outside of the turn of a `Piece`:
        - Within the function that makes a `Piece` defend against damage:
            - Number to check against dodge chance; then
            - Number to check against the parry chance.
    - Within the turn of a `Piece`:
        - Within the function that makes a `Piece` move – specifically the Hammer and the Runner as they are the only two that require it:
            - Number to check for where to move next in randomly moving around the initial position.
        - Within the function that makes a `Piece` attack:
            - Number to check against hit chance; then
            - Number to check against the critical hit chance.

  There should be no other time in which the random function is called.
- A class has been provided with all the output messages that will be needed. This will prevent the loss of marks due to spelling mistakes. For the messages that do not end, such as `ACTION_ATTACK` and `GAME_END_COMPLETE`, the only addition is the relevant integer values, followed by the message `SENTENCE_END`.
- At the beginning of Phase 2, you will be given the memo code for Phase 1 and at the beginning of Phase 3, you will be given the memo code for Phase 2. This is done so that if you were unable to complete the previous phase 100%, you still have a chance to complete the current phase. Under the code folder is another folder named *html*. If you navigate into this folder and open the *index.html* file,

you will find Doxygen generated documentation which you may use in understanding the given code.