ATAR Computer Science

The Smith's Stash

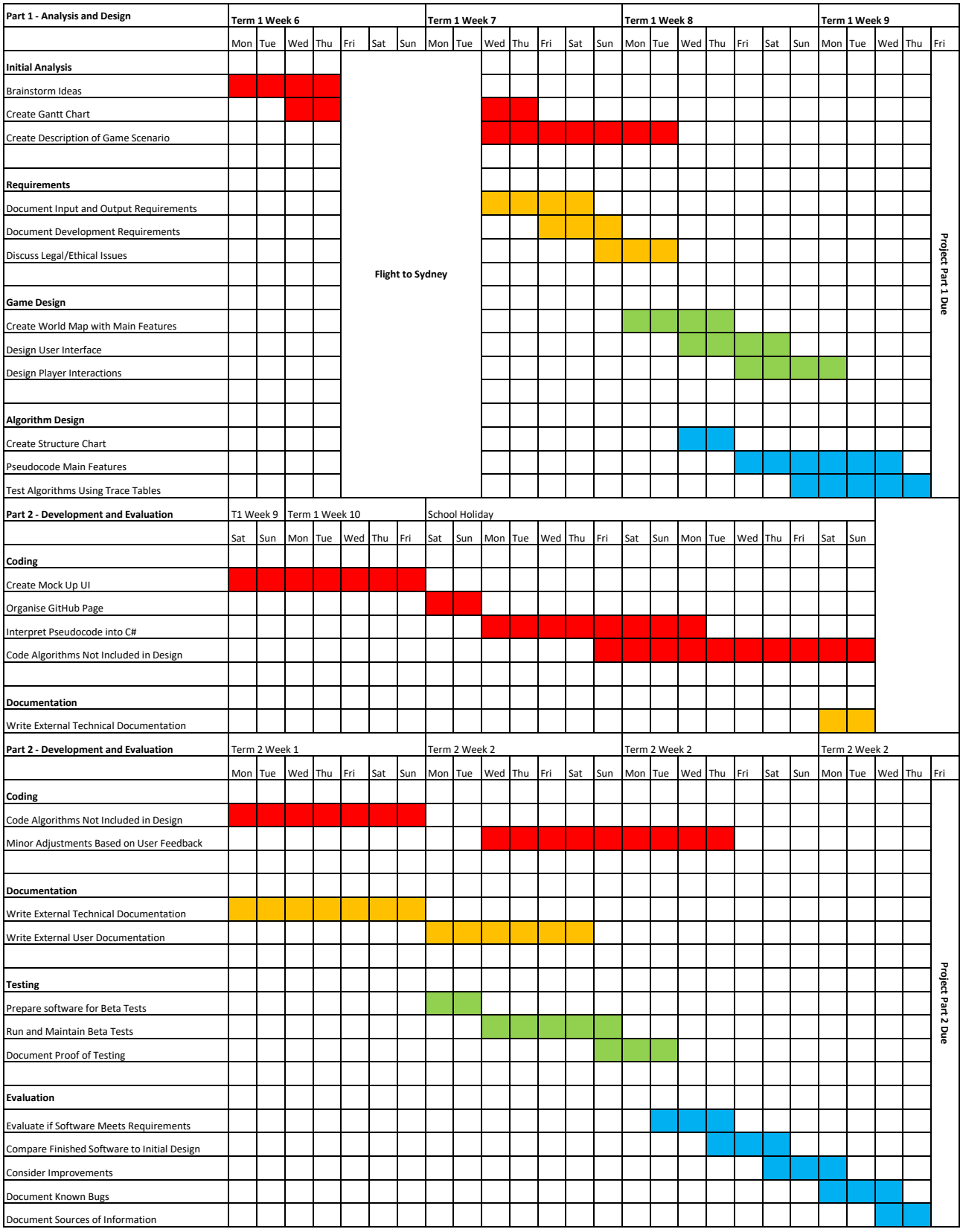
Project 1 Part 1 – Analysis and Design

Alistair Parkinson
4-5-2019

Contents

Pre-Analysis

Gantt Chart



Game Scenario

Background Plot

The world in which the game is based is of the high fantasy genre. It is largely inspired by the works of J.R.R. Tolkien and other role-playing games, such as Dungeons and Dragons. The technology of the time is similar to that of towns of medieval era. There are two main towns in which the story is based, Bardford and Sharnwick, which are joined together by a trail, notorious for its ambushes.

Role of Player

The player adopts the role of an ordinary townsperson from the town of Bardford, who has been out of work for a long time and is looking for a job.

Game Objectives

The objective at the beginning of the game is to meet your Dwarven friend, Baern, in the town of Sharnwick, but after learning that he had been ambushed, your objective is to find and rescue him.

Synopsis

The game begins in the town of Bardford, when the player receives a letter from a Dwarven friend Baern Rockseeker. He requests that you meet him in the town of Sharnwick as he wants you to help him with his new quarry business. In the envelope is 10 gold pieces (the currency of the game world), which he recommends you spend at the armoury, and he warns you that the trails are notorious for their ambushes. During your walk to Sharnwick, you stumble across a dead horse that had been shot dead by an arrow. Reading the name on the emptied saddlebag, you learn that it belonged to Baern (your friend). When you are ambushed by the same goblins who ambushed Baern, you use the items you bought from the armoury to rescue Baern and escape.

Requirements

Development Factors

User Needs

The basic of the task is to create a text-based role-playing game. Role playing games generally involve the player assuming the role of a character in a fictional setting. To achieve this, the player must be able to control the movement and actions of their character. If the player feels they cannot freely control their character, it could break immersion, thus making the player less engaged in the game. Since in the end, the task is to create a computer game, the success is dependent on whether the game is fun and engaging for the user.

The criteria that I have come up with to judge how well the software meets user needs is the following:

- The user must feel immersed in the world and character that they play, thus the player should feel a sense of control over their character and that their decisions impact the world in which the game is based.
- The game keeps the user engaged and strikes a balance between being challenging and rewarding
- The game should be reliable and bug free as not to break immersion.

Apart from that, further testing will be necessary to further determine the user needs.

Potential Users

Apart from the marker, people that may use my software will most likely source my software from GitHub. To cater to this audience, I need to document my GitHub page in a way in which makes it easy for a user to navigate. This includes:

- Separating working code from broken code
- Warn users of potentially damaging code
- Internally documenting my software well
- Make installation process user friendly

User Interface

I am restricted to making a text-based adventure, in the form of either a C# WPF or console application. Console applications run in a terminal window and can daunting for a user who is not familiar with the interface. WPF Applications on the other hand are generally more user-friendly than console applications as they can make use of on-screen buttons and menus.

Because the software is developed as a C# app using Visual Studio. It means that the UI must run on a desktop or laptop device with Windows installed. Therefor I must ensure that my game is compatible with a variety of different monitor sizes and aspect ratios, the most common being 5:4, 4:3, 16:10 and 16:9.

Because the game is a text adventure, it is likely that user will want to play it in a window rather than taking up an entire monitor, thus I should design the game to work for a variety of different aspect ratios and also make sure that UI is not too cluttered so that it appears nicer in a small window.

I have identified the modes of input as the keyboard and mouse/trackpad, and the modes of output as the monitor. Due to the limited support for trackpads and mice in console applications, I must design an interface that only requires keyboard input.

I should make sure to minimise the amount of input needed per action from the user, as to increase the efficiency of interaction between the user and the software.

I should handle input errors in a way that the input error is clear to the user but does not cause the user annoyance.

I should output only what adds to the player experience as constantly outputting unnecessary information could make the game window cluttered and take away from the user experience.

Finally, I need to make the controls intuitive so that a user who has not played a text RPG before will be familiar with the interface and will not need to refer to external user documentation.

Processing Efficiency

The text RPG will most likely not suffer from performance issues due to it being text based. Text based applications tend to be easier to process than applications with complex UIs. Additionally, since this program will be run on laptops and desktops (as explained in previous section) as opposed to less powerful mobile devices such as phones and tablets, thus the software will run faster. I will therefore be able to produce algorithms that are more complex without sacrificing performance. To improve processing efficiency, I can increase the speed that the user interacts with my software by focusing on making the user interface quicker and easier to user.

Licencing Requirements

Software Licensing is usually used to protect the source code and limit the distribution of the software, for commercial purposes.

However, since this is merely a school project, I do not plan on making commercial gain from my software. If I choose not to use a license, my work is under exclusive copyright by default, meaning that, I have the right to file take-downs, on anyone who copies, distributes, or modifies my work.

Open-source licenses allow software to be freely used, modified, and shared, whilst usually still maintaining certain rights of the owner. I have decided that as a requirement I will need to use some sort of open source license.

The following license below is the template for the MIT license that I must include in the source code of my software, if I am to use it:

MIT License

Copyright (c) [year] [fullname]

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

In summary, the license allows the commercial use, distribution, modification and private use of the software and source code, given that I (the owner) am credited and that the copy of original MIT license is included with the distribution code.

Additionally, I am protected if in any circumstance I am held liable to any claims, damages or other liabilities.

Legal Obligations

I have made sure that my software coincides the following government acts.

The Copyright Act (1968) and Copyright Amendment (Digital Agenda) Act (2000)

The Copyright Act protects owner's intellectual property from being unfairly used by others without the owner's consent. Intellectual property includes source code, and algorithms so I need to make sure that I am not using other people's source code when developing, unless I am given permission to do so. I have evaluated that basing the world off the world of dungeons and dragons, is not an infringement of the copyright act as the company Wizards of the Coast, does not own the intellectual property to dwarves, elves, goblins, etc.

Privacy Act (1988) and Privacy Act (2000)

The Privacy Act (1988) ensures that personal data from users that is stored and collected by software is used in a truthful and lawful manner.

Because the collection of personal data will not affect the gameplay of text RPG, it would make sense not to store personal data, due to the risk of violating the Privacy Act, if the data is not secured well enough.

If I were to collect personal information, I must ensure that it coincides with the 11 Information Privacy Principles as outlined in the Privacy Act.

Spam Act (2003)

The Spam Act disallows the sending of spam, which is define as the sending of unsolicited commercial electronic messages, such as SMS and IM. It also requires the organisation give you the option to users so that they can removed from the companies mailing list. It is illegal to sell email address lists. And finally, I cannot use software to search the internet and compile an email address list. It is unlikely that I violate any of these conditions, given that the software will run completely on the client side and will not require internet connection, thus I will not be able to collect user data.

Ethical Obligations

I need to make sure when designing and developing my software that it is ethical. Ethics are not necessarily required by law, but instead guided by moral principles and virtuous acts, meaning that one person's code of ethics may not be the same as another's. Below I have listed my own code of ethics which I will follow during the design and development of my project.

- My software must be safe to use and not contain security flaws, even though I do not hold liability if something does happen.
- My software must not purposely offend any of my users.
- My software should not unfairly take advantage over its users.

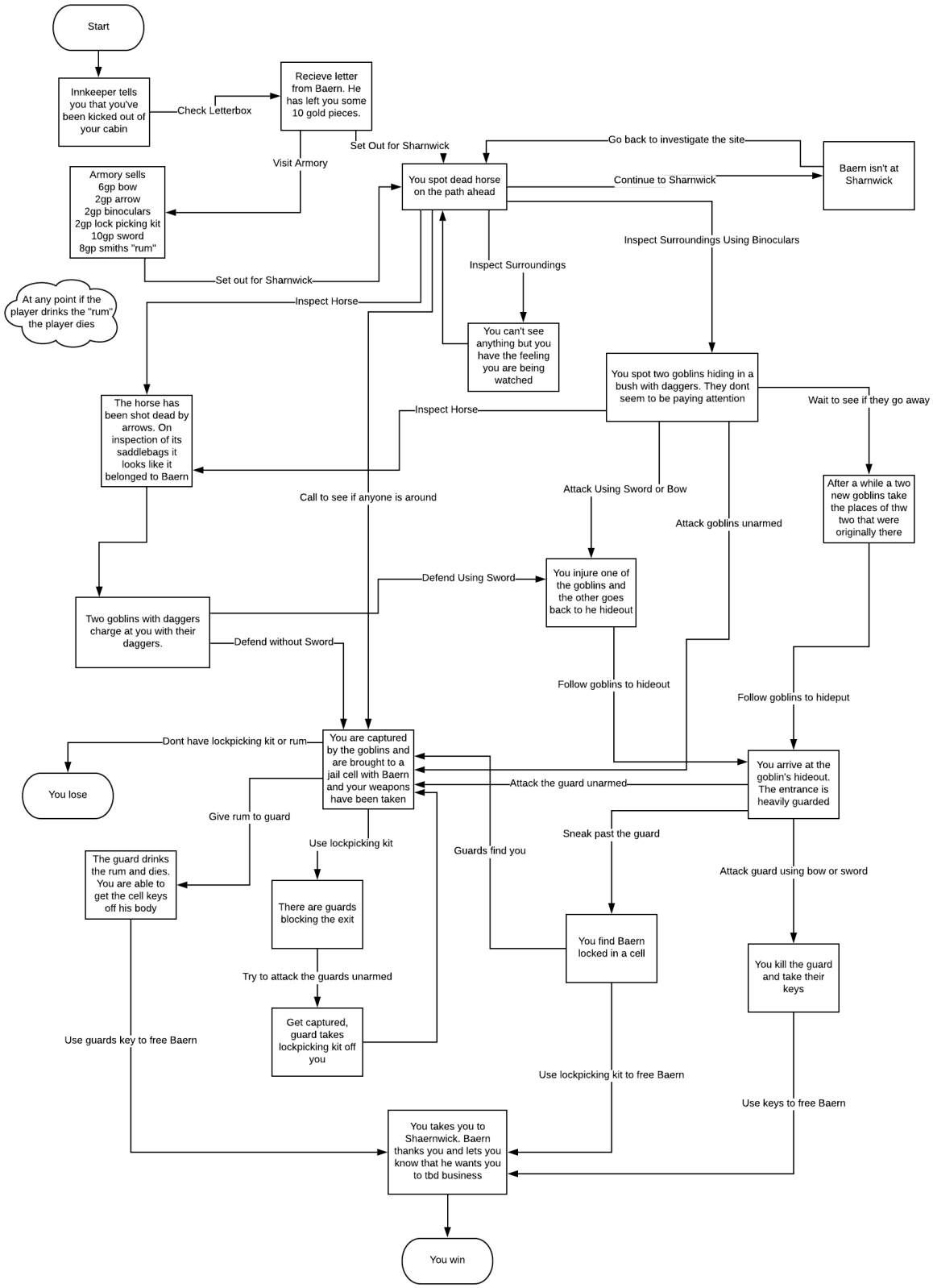
To ensure that my software is ethical, I will use the general code of ethics from the Australian Computer Society, to guide the design of my software in terms of ethics. This general code of ethics includes the following.

Primacy of public interest	Nothing in the text-based RPG is likely cause a detriment to society.
Enhancement of quality of life	I need to ensure that I am not purposely wasting the time or causing detriment to the users or my users.

Honesty	I need to ensure that when I promise my users is a text role playing game, it is no more than a text role playing game.
Competence	I need to ensure that the code I distribute is free of bugs, as not to waste users time.
Professional development	Because I am using an open source license, others will be able to learn from my code, thus furthering their professional development.
Professionalism	I need to lead by example, by upholding principals of ethics and law.

Game Design

Flow Chart



User Interface

Of the two options I have, to create the user interface with, I have chosen to use the console application, as I am more familiar with the C# console applications than WPF applications. Whilst this could have a negative effect on the user friendliness of the UI to people unfamiliar with console applications, I feel that the minor benefits it provides to the user experience are outweighed by the time it may take to develop and the possible bugs that could arise.

I have decided the user interface is to be designed similar way to that of text adventures such as:

- Zork
- The Hobbit
- The Hitchhikers Guide to the Galaxy Game

In these games, your location and scenario is printed to the console and you are prompted to type what your character does next. Often these games are able recognise simple actions such as “turn on light”, “attack goblin” or “pickup toothbrush”, and will provide appropriate feedback to the user, which is printed to the user.

From playtesting I have noticed that the game often restricts what the player can do, thus often leading to user frustration, and the need for tutorials. Because the player in my game is often limited to 2-3 actions per scene, choosing to interpret the actions the user is intends to do could be a problem.

To solve this issue designed the UI is to provide a list of possible actions rather than letting the user decide the exact action they want to do. A disadvantage of this is that the user may feel restricted, in terms of control of their own character. Below is a potential layout of my user interface.

```
You've been on the trail to Sharnwick for about half a
day. As you come around a bend, you spot a dead horse
sprawled about fifty feet ahead of you is, blocking the
path. It several black-feathered arrows sticking out of
it. The woods press close to the trail here, with a steep
embankment and dense thickets on either side.

What do you do next?

A – Inspect the horse

B – Use your binoculars to scan the area for danger

C – Call and see if anyone responds

D – Go around the horse and continue journey to Sharnwick

> [user types A, B, C or D]
```

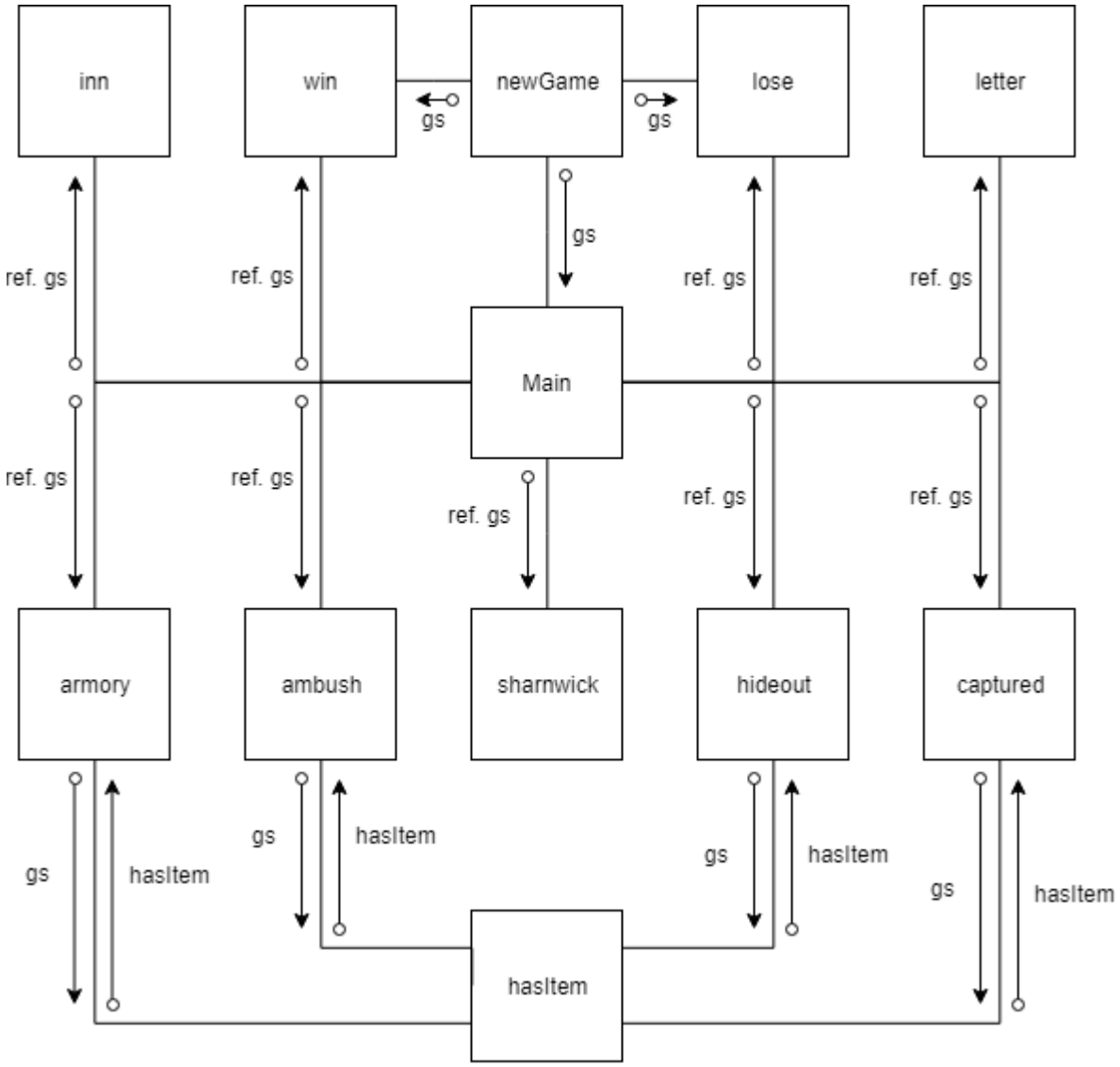
I have concluded that this interface meets the following requirements as outlined in the analysis stage:

- The user interface is initiative and does not necessarily need external user documentation or previous experience in playing text adventures

- The user interface is scalable to different monitor and window sizes (as long as user changes the console font size appropriately)
- The input is minimised as only two keyboard presses are required per action (The letter and the enter key)
- The user interface handles input errors in a way that will not confuse or frustrate users (user interface simply prompts the user to re-type input)
- No unnecessary output is printed to the screen as it does not print out the game state every action but instead flavourful text to aid in the immersivity of the game.

Algorithm Design

Structure Chart



Algorithm Overview

The game state, or gs as written in the code and the dfd, is a record that outlines all of the data required to describe the current state in the game. It includes:

- Key locations that the player has visited
- Significant events the player has triggered
- The items that the user is holding
- The current area that the player is at / process that will run next

The basis of how this algorithm works is that each different place in the game has its own procedure. Each procedure is parsed a reference parameter to the current game state. The procedure will in turn make appropriate changes to the game state including, updating the players inventory and current area. The main advantage of this design is that it is modular. Processes can be tested

individually, can be replaced with stubs when necessary and helps with the process of finding and debugging errors, as they can easily be pinpointed.

Pseudocode

```
MODULE Main
	gs = newGame()
	WHILE (gs.currentArea != “quit”)
		CASE gs.currentArea OF
			=”inn”: inn(gs)
			=”letter”: letter(gs)
			=”armoury”: armoury(gs)
			=”ambush”: ambush(gs)
			=”sharnwick”: sharnwick(gs)
			=”captured”: captured(gs)
			=”hideout”: hideout(gs)
			=”win”: win(gs)
			=”lose”: lose(gs)
			default: ASSERT // crash code to prevent infinite loop
		END CASE
	END WHILE
END Main

RECORD GameState
	beenToSharnwick		// bool
	seenArmory		// bool
	inspectedSurroundings	// bool
	inventory		// list of strings
	gp			// int
	currentArea		// string
END RECORD

FUNCTION newGame()
```

```
	gs <- new GameState
	beenToSharnwick <- false
	seenArmory <- false
	inspectedSurroundings <- false
	gs.inventory <- new list
	gs.gp <- 0
	gs.currentArea <- ”inn”
	RETURN gs
END newGame


FUNCTION hasItem(item, gs)
	output = false
	FOR i <- 0 TO gs.inventory.length
		IF item == gs.inventory[i]
			output = true
			BREAK
		END IF
	END FOR
	return output
END hasItem


PROCEDURE inn(ref. gs)
	PRINT(“You hand your keys over to the innkeeper. You’ll need find somewhere else to stay.”)
	PRINT(“As you exit the door the innkeeper reminds you “Your friend Baern left a message for you, take this”, the innkeeper hands you a letter”)
	gs.inventory.add(“letter”)
	in <- 0
	DO
		PRINT(“Press ENTER to read the letter”)
		in <- READ
	WHILE in != ENTER
```

```
END inn


PROCEDURE letter(ref. gs)
      in <- READ
      PRINT("You open the letter, it reads as follows...")
      PRINT("You decide that you have no choice other then but to set out for Sharnwick, but then you remember what he said about the trails being dangerous.")
      in <- 0
      DO
            PRINT("What do you do next:")
            PRINT("A – I'll head to over the town's armoury.")
            PRINT("B – I don't need that greedy smith wasting my time and money. I'll just leave now.")
            in <- READ
      WHILE in != A AND in != B
      CASE in OF
            ="A": gs.current <- "armoury"
            ="B": gs.current <- "ambush"
      END CASE
END letter


PROCEDURE armory(ref. armory)
      IF seenArmoury = FALSE
             PRINT("Description of armoury")
             seenArmoury <- TRUE
      DO
             PRINT("What do you do next:")
             PRINT("A – purchase a 10gp sword")
             PRINT("B – purchase a 7gp bow")
             PRINT("C – purchase a 1gp arrow")
             PRINT("D – purchase a 2gp lockpicking kit")
             PRINT("E – purchase a 2gp pair of binoculars")
```

```
            PRINT(“F – purchase a 8gp bottle of the smith’s “run””)
            PRINT(“G – set out to Sharnwick”)
            in <- READ
      WHILE in != A AND in != B AND in != C AND in != D AND in != E AND in
!= F AND in != G
      CASE in OF
            =”A”:
            IF gs.gp >= 10
                  PRINT(“You purchase the sword”)
                  gs.inventory.add(“sword”)
                  gs.gp -> gs.gp - 10
            ELSE
                  PRINT(“You do not have enough gp to purchase that”)
            END IF
            =”B”:
            IF gs.gp >= 7
                  PRINT(“You purchase the bow”)
                  gs.inventory.add(“bow”)
                  gs.gp -> gs.gp - 7
            ELSE
                  PRINT(“You do not have enough gp to purchase that”)
            END IF
            =”C”:
            IF gs.gp >= 1
                  PRINT(“You purchase the arrow”)
                  gs.inventory.add(“arrow”)
                  gs.gp -> gs.gp - 1
            ELSE
                  PRINT(“You do not have enough gp to purchase that”)
            END IF
            =”D”:
            IF gs.gp >= 2
```

```
                PRINT("You purchase the lockpicking kit")
                gs.inventory.add("lockpicking kit")
                gs.gp -> gs.gp - 2
            ELSE
                PRINT("You do not have enough gp to purchase that")
            END IF
            ="E":
            IF gs.gp >= 2
                PRINT("You purchase the binoculars")
                gs.inventory.add("binoculars")
                gs.gp -> gs.gp - 2
            ELSE
                PRINT("You do not have enough gp to purchase that")
            END IF
            ="F":
            IF gs.gp >= 8
                PRINT("You purchase the rum")
                gs.inventory.add("rum")
                gs.gp -> gs.gp - 8
            ELSE
                PRINT("You do not have enough gp to purchase that")
            END IF
            ="G": gs.currentArea <- "ambush"
    END CASE
END armory

// stub
PROCEDURE ambush(ref. gs)
    gs.next = "win"
END ambush

// stub
```

```
PROCEDURE sharnwick(ref. gs)
	gs.next = “win”
END sharnwick

// stub
PROCEDURE hideout(ref. gs)
	gs.next = “win”
END hideout

// stub
PROCEDURE captured(ref. gs)
	gs.next = “win”
END captured

PROCEDURE win(ref. gs)
	PRINT(“Congratulations, you have won the game! Would you like the play again? (Y/N)”)
	In <- READ
	WHILE in != “Y” AND in != “N”
		PRINT(“Would you like to play again?”)
	END WHILE
	IF in == “Y”
		gs <- newGame()
	ELSE
		gs.current <- “quit”
	END ELSE
END win

PROCEDURE over(ref. gs)
	PRINT(“You Lost!”)
	in <- 0
	DO
```

```
            PRINT (“Would you like to have another try? (Y/N)”)
            in <- READ
      WHILE in != “Y” AND in != “N”
            gs <- newGame()
      ELSE
            gs.current <- “quit”
      END ELSE
END over
```

Desk Checking Main Module

Test Data is: X -> ENTER -> Y -> A -> Z -> B -> F -> G -> Y -> ENTER -> B -> Y

in	gs.seenArmory	gs.gp	gs.inventory	gs.currentArea	Shortened Output
0	FALSE	0	EMPTY	“inn”	You hand over the keys… What do you do next?...
W			“letter”		What do you do next?...
ENTER		10		“letter”	You open the letter… What do you do next?...
0					
X					What do you do next?
A	TRUE			“armoury”	Description of armoury… What do you do next?...
Y					What do you do next?...
B		3	“letter”, “bow”		You purchase the bow
F		3			You do not have enough gp to purchase that
G				“ambush”	
Z				“win”	Congratulations, you have won the game! Would you like the play again? (Y/N)
Y	FALSE	0	EMPTY	“inn”	Would you like the play again? (Y/N)
0			“letter”		You hand over the keys… What do you do next?...
ENTER		10		“letter”	You open the letter… What do you do next?
0					
B				“ambush”	
Y				“win”	Congratulations, you have won the game! Would you like the play again? (Y/N)
				“quit”	

Main Function Behaves as expected.