Software Requirements Specification for the Dragon Adventure Game

Document # CSC444-SRS-001A

Revision A

8th September 2002

Table of Contents

1 Scope

This specification establishes the functional, performance, and development requirements for Release 1 of the Dragon Adventure Game Software.

1.1 Purpose

The Dragon Adventure Game is an interactive computer game with a textual interface, in which the user explores a series of interconnected rooms, collecting artifacts, and fighting monsters. The set of rooms, artifacts and monsters can be extended or replaced to give different game variations.

1.2 Definitions, Acronyms and Abbreviations

CSCI	Computer Software Configuration Item
SRS	Software Requirements Specification
DM	Dialogue Management CSCI
MP	Mapping CSCI
AM	Artifact Management CSCI
CFM	Creature and Fight Management CSCI

1.3 Reference Documents

The following standards apply

DOD-STD-498A	US Department of Defence Software Documentation Standard
J-STD-016-1995	IEEE/EIA Standard for Information Technology, Software Lifecycle Processes, Software Development, Acquirer-Supplier Agreement
IEEE-STD-P1063	IEEE Standard for Software User Documentation

The following documents describe the course in which this software is to be developed:

CSC444-HND-001	Course Orientation Handout
CSC444-HND-002	Notes on the Software Trading Game
CSC444-ASG-001	Content description for Assignment 1

1.4 Document Overview

Section 1 identifies the scope of this document, the purpose of the software, and lists the definitions, acronyms and reference documents. Section 2 lists the documents referred to elsewhere in this document. Section 3 identifies the four main Computer Software Configuration Items (CSCIs) that comprise the system, and gives the functional requirements and constraints for each CSCI. Section 3 also describes the quality requirements for the software.

2 Overall Description

2.1 System Perspective

Back in the days before graphical user interfaces became popular, a number of entirely text-based games were written, the most famous of which was "Adventure" which was found on most UNIX systems. In this game, the user "roamed" through a series of interconnected rooms (or rather, caves) by conducting a textual dialogue with the machine. The machine would describe what the user could "see", and the user typed in various commands which were a simple subset of English language sentences. Each command contained one of a limited number of verbs (e.g. go, open, hit, look, grasp, throw), together with an object and/or direction. During the game, the user would collect various artifacts, such as weapons, treasure and magical potions, and fight the monsters he or she encountered. This particular game has entered the computing folk-law, mainly for the phrase "you are in a maze of twisty passages all alike" as there was one portion of the game from which it was virtually impossible to escape. Part of the difficulty of the game arose from remembering where you were, and where you had been, as the computer offered no map of the rooms. Drawing your own map helped, but was not easy, as the interconnectivity of the rooms was often a little complicated.

This specification is essentially a re-implementation of the Adventure game. As with the original adventure game, the game specified here is single user only. The specification is, of course, open ended, in that an unlimited set of rooms, artifacts, and monsters can be added.

2.2 System Functions

There are essentially four main functional areas, which correspond to the four CSCIs specified in section 3:

- Dialogue Management: The system uses a simple command line interface, with a simplified natural language interface. A limited vocabulary is used, but the vocabulary is defined *a priori*. Interaction proceeds between the user and the game proceeds as a dialogue, in which the user enters a command, and the system describes a response to the command. Users are shown some sample commands, and may try creating further commands by guessing the vocabulary.
- Mapping: At the beginning of the game, a map is set up of all the rooms in the game, the doors that connect the rooms, and directions needed to move from room to room. A short and long description is maintained of each room, and each door; the user can access these by using various exploration commands. The mapping functions also include keeping track of which artifacts and monsters are in which room. The contents of each room and the descriptions of monsters, objects and rooms are updated during the game, for instance during fights, or when the player picks up or drops things.
- Artifact Management. The player can find and collect a number of artifacts during a game. These include weapons, food, tools, treasure and so on. Different artifacts can be used for different things. The artifact management functions include keeping track of descriptions of artifacts, and of what can be done with them. For example, some artifacts are tools that can be used on other artifacts (e.g. a key to a locked box).
- Creature and Fight Management: When a player encounters monsters during the game, a fight can ensue. Players can employ a number of different weapons to attack monsters, and monsters may attack back. If the monster wounds the player, points are deducted from the player's current strength. The game ends if the players' strength reaches zero. Monsters may also pursue users as they move from room to room, and may attack one another.

2.2.1 Class hierarchy

Figure 2.1 shows the class hierarchy for the game. This kind of diagram shows the classes of objects that need to be represented in the game, and the attributes that each class of object has. Object classes are shown in bold, and the attributes are shown in italic. Note that there is full inheritance, which means that objects inherit all the attributes of their parents. For example, every object has a unique id and a description, whereas only moveable objects have short names.

2.3 User Characteristics

The game should be useable by any users, via a command line interface. No special knowledge or skills should be assumed on the part of the users. Users are should not be expected to learn a set of commands in order to start using the game.

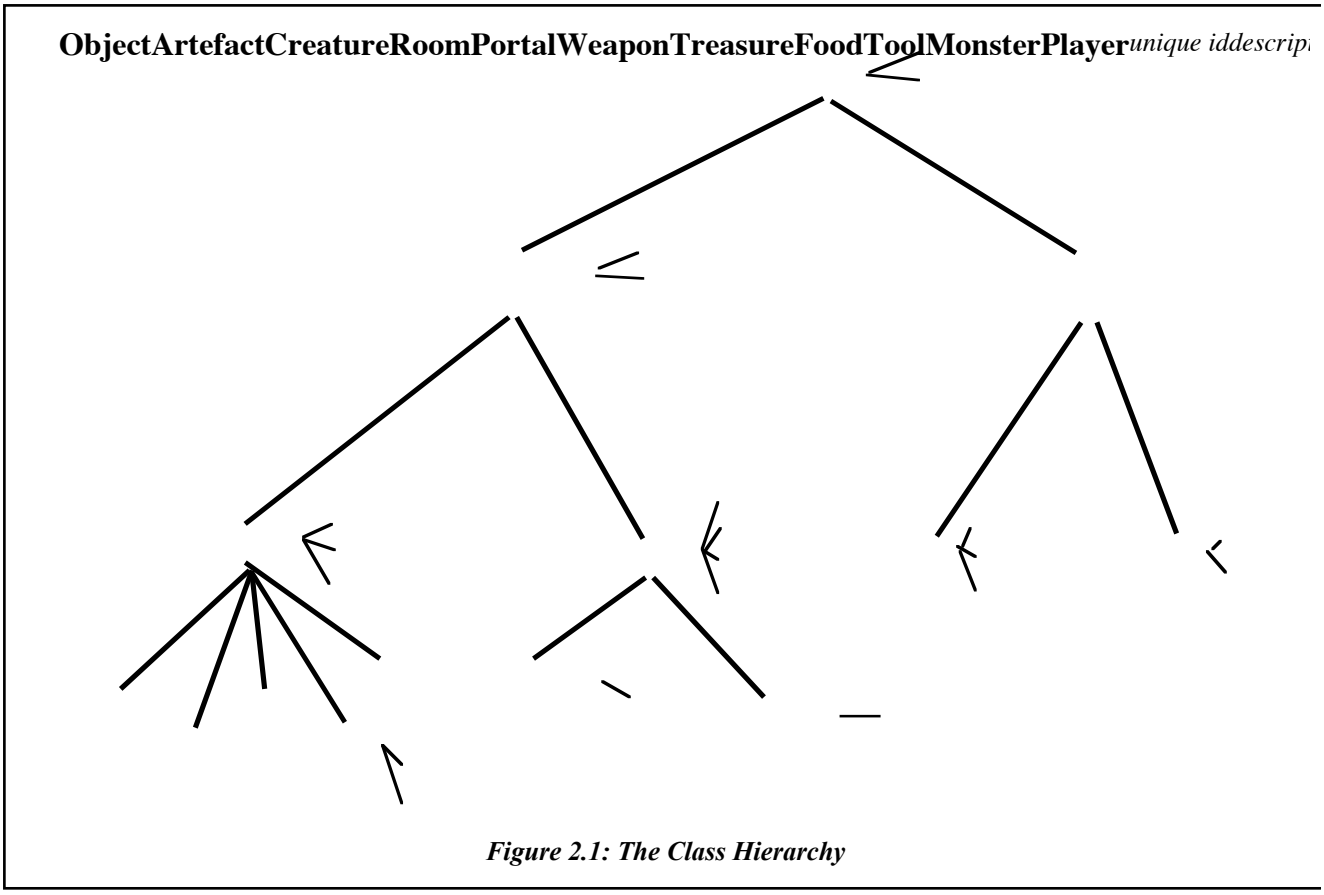

Figure 2.1: The Class Hierarchy

2.4 Constraints

No special constraints have been identified.

2.5 Assumptions and Dependencies

No special assumptions or dependencies have been identified.

3 Overview

3.1 External Interface Requirements

3.1.1 User Interfaces

All interaction with the user is via a command line interface. Once the game has started, the user is prompted for a command. The game prints out a response and prompts for the next command. If the user enters a command that is badly formed, the system shall report that the command was not understood.

3.1.2 Hardware Interfaces

None

3.1.3 Software Interfaces

The system shall be capable of running on any version of UNIX system, including Linux. The system shall make use of the operating system calls to the file management system to store and retrieve game states.

3.1.4 Communication Interfaces

None

3.2 Functional Requirements

The functional requirements are described for each of the four CSCIs: Dialogue Management (DM), Mapping (MP), Artifact Management (AM), and Creature and Fight Management (CFM).

The management of data (i.e. the responsibility for representing information about objects during the game) is distributed between the MP, AM and CFM. MP is responsible for storing all data concerned with fixed objects, including the location of all the moveable objects (these are allocated to rooms at initialization). AM is responsible for all artifacts, including their descriptions and usage. CFM is responsible for storing all creature data. Note that the player is a creature, and therefore CFM will be responsible for keeping track of information about the player (i.e. how much strength she has left, how many points she has accumulated, etc.).

3.2.1 Dialogue Management CSCI (DM)

The dialogue management CSCI initializes the game, and controls the dialogue with the user. This CSCI does not store any data about the creature, artifacts and players during the game. It calls functions from the other CSCIs as and when they are needed.

DM is the main controlling software for the game. It implements a cycle of providing a message to the user, and asking for a command. It processes the user's commands, and performs the appropriate actions. It constructs all messages and descriptions given to the user, based on information provided by the other CSCIs. It provides appropriate feedback if the user types an invalid command. It calculates scores when asked, and can save and restore game states.

3.2.1.1 DM Internal Data

DM has to perform a certain amount of parsing which will be facilitated by constructing the following lists of verbs:

- movement verbs (e.g. go, move, walk, etc.) – these are used with a direction to move between rooms. The direction must be one of those supplied as a valid portal from the current room.
- fight words (e.g. hit, attack, etc.) – these are used to attack a monster, using the current weapon, if any. If the player is not currently grasping a weapon, then bare fists are used, which have a strength of 1. If there is more than one monster in the room, then the fight word must be used with the short name of the monster to be attacked. Otherwise, the fight word can be used on its own or with 'it'.
- information words (e.g. look, inspect, etc.) – these are used to get detailed descriptions. Used on their own, they apply to the room. If combined with a direction, they get the description of the portal in that direction. If used with a moveable object, they get the full description of that object.
- inventory words (e.g. put, get, pick up, drop, grasp, wield, list, etc.) – these are used to manipulate artifacts. The player may pick up any artifact in the current room, by giving the appropriate verb, and the artifact's short name. Artifacts picked up are removed from the room, and added to the player's artifact list. Similarly, items may be put down. Any artifact that has been picked up may be used as a weapon by using

the verb wield, plus the artifact's short name. Artifacts that are not of the weapon class are ineffective as weapons (i.e. they do no damage to any monsters). The player can only wield one artifact at a time; wielding a new artifact implies the previous weapon is unwielded; although both are still in the player's artifact list. The verb 'list' can be used to list the short names of all artifact's carried by the player.

- action words – These are all the verbs that can be used on specific objects, or classes of objects. This list must be compiled from the various objects used in the game, and is generated by a function provided by AM. These words must be combined with an object, to perform an action on that object. Tool use words are a special subset of action words. These require both an object and a tool. E.g. "unlock the oak door with the brass key"; "hit the egg with the hammer", etc.). It may be desirable to break this into two interactions, e.g. "hit the egg" to which the system responds "with what?"...
- game commands (e.g. save, restore). These commands are used to save the current state of the game, or to restore a previous game from a file. The name of a file is required as a parameter.

In addition to these commands, the user can always give the command 'help' which provides a limited amount of help about how to play the game. This help message should also be given at the beginning of the game.

DM does not keep track of information about the player. The player always has unique id 0. The location and other attributes of the player can be determined using the locate(creature) command in CFM, with unique id 0.

3.2.1.2 DM Control Structure

After initialization of the rooms and things, the program describes the first room, and then enters the main loop. The main loop carries out the following steps:

1. The system prompts the user for input
2. The user types in a command (usually in the form verb + object, e.g. "go north")
3. The user's command is parsed, and the verb and object are checked to see if they are on the master list:
   a) if the verb is a movement word, and the direction is a valid portal from the current room, it moves the player into a new room. The user gets a full description of the room and its portals, and the short names of any moveable objects in the room.
   b) if the verb is a fight word, and the named monster is in the current room, then the monster is attacked. If no monster is specified, and there is only one monster in the room, then this monster is attacked. CFM handles the fighting.
   c) if the verb is an information word, then a description is output. If the word is used alone, then a description of the current room, plus its exits is given. If it is used with a direction, the portal (if any) in that direction is described. If it is used with an artifact, then the full description of that artifact is given.
   d) if the verb is an inventory word, and the given action can be performed, then the named artifact is added/removed from the player's artifact list; or is wielded as a weapon; or the inventory is listed.
   e) if the verb is an action word, and is used with an artifact, then the artifact is checked to see if it is a valid action. If it is a valid action, the action is performed on the artifact. The result is described to the user. If the command is an eat action, and the artifact is food, then the strength of the artifact is added to the player's strength.
4. The fight and move functions of CFM are called to see if any monsters attack the player, or move from room to room. A description of any fighting and it's result is given to the user.
5. If the player is dead the game ends. Otherwise the loop continues.

3.2.2 Mapping CSCI (MP)

The mapping CSCI maintains a list of rooms and connections between them, and keeps track of the things contained in each room. It also provides a textual description of each room when needed.

3.2.2.1 MP Internal Data

Each room is represented within this CSCI using the following information:

- a unique id – Each room has a different number. This information is never divulged to the user. The player will always start in room 0.
- a description – this is the textual description given to the user upon entering the room, or upon giving the 'look' command. The description should not contain any information about the exits, nor any removable objects or monsters, as these are described separately. It should provide information about furniture and

fixed objects.
E.g. **Description:** *[a huge banqueting hall with chandeliers hanging from the ceiling and a very long oak table capable of seating fifty or so guests, but there are no chairs. There is no other furniture in the room. The chandeliers are not lit, and the only light comes from a small barred window set high into the south wall]*

- a portal list – each exit from the room is represented by a unique id, a description, a direction and a successor, which is the room number of the connecting room. The direction allows the user to specify which exit she wishes to go through, and the description is provided merely to help the user visualize and remember the scene.
  E.g. For each portal:
  Unique id: *[Door 57]*
  Description: *[a heavy wooden door with large iron hinges set high into the wall]*
  Direction: *[north]*
  Successor: *[42]*.
  The user will be presented with the text "To the north, there is a heavy wooden door with ...", and might then issue the command "go north".
- an artifact list – the list of artifacts in the room, represented by their unique ids.
- a creature list – a list of creatures in the room, represented by their unique ids.

Portals need not be represented separately.

3.2.2.2 MP External Interface

MP will provide the following functions to be used by other CSCIs:

- A successor function: Given the unique id of a room, and a direction, returns the unique id of the next adjacent room in that direction.
- A rooms initializer: Given a set of moveable objects, represented as a list of unique ids, randomly places the things in various rooms, and returns an ordered list of the unique ids of the rooms the moveable objects were placed in.
- A room describer: Given the unique id of a room, returns a textual description of that room.
- A room decorator: Given the unique id of a room, and a piece of text, adds the text to the end of the description of the room. This function is used for permanent changes to a room (e.g. broken furniture and fittings, bloodstains, etc).
- A portal lister: Given the unique id of a room, provides a list containing the direction and description of every portal in the room.
- An artifact lister: Given the unique id of a room, provides a list of unique ids of all artifacts in that room. Note this list will not include artifacts carried by any monsters in the room.
- A creatures lister: Given the unique id of a room, provides a list of unique ids of all creatures in that room.
- An add moveable object function: Given the unique id of an object, and its class (creature or artifact) and the unique id of a room, adds the object to the relevant list of things in that room.
- A remove moveable object function: Given the unique id of an object, and its class (creature or artifact) and the unique id of a room, removes the object from the relevant list of things in that room.

3.2.3 Artifact Management CSCI (AM)

AM provides data structures to hold information about each artifact. It initializes and updates the information about artifacts during the game. It also keeps track of what actions can be performed on which type of artifacts. Artifacts can be picked up by the player, and carried from room to room. They can also be used for various purposes, and using them sometimes destroys them.

3.2.3.1 AM Internal Data

Artifacts are represented with the following information:

- A unique id – used to refer to the artifact. This information is never divulged to the user.
- A short name – used to refer to the object when interacting with the user. This should be long enough to identify the object, but short enough to allow the user to type it in easily. It is possible to have more than one object with the same short name.

- A description – a detailed description of the artefact. When the player enters a room, she will be given the short names of any artefacts found there ("there is a short bow, a cloth bag and a red gem here)". If she looks at an object specifically ("look gem") the detailed description will be given. The detailed description may be empty, in which case the user will get a message such as "it is an ordinary red gem".
- A type – An artifact may be one of: food; weapon; treasure; tool, thing. Food may be eaten to gain strength, weapons may be wielded to improve fighting ability, treasure adds to the player's points; tools are artifacts that operate on other objects, things are simple artifacts and are usually of no particular relevance to the game.
- A list of actions – these are the actions that can be performed on the artifact, represented as a verb, and a message. These are in addition to the actions that can be performed on any artifact (such as pick up, put down, throw) or on all of a particular type of artifact (e.g. food always has the action eat). The message is given to the user if she performs the action. If the message is empty the user will get a message of the form "nothing happens".
  E.g. if the artifact is "red gem", an action might be
  Verb: *[rub]*
  Message: *[the gem shines a little more brightly]*.
- Usage – A positive integer representing the number of times the artifact can be used. If this value reaches zero, the artifact is destroyed. For example, a banana can only be eaten once, but a bunch of bananas could be used, say, six times. This number is decremented each time the artifact is used. The special value -1 is used for artifact that can be used any number of times (e.g. most weapons, and all treasure can be used any number of times).
- Strength – An integer representing the strength of the artifact (the default is 1). E.g. for food, this is the number of strength points gained by eating the artifact, for a weapon this is a multiplier for determining how much damage is done. For treasure, this is its financial value (in gold pieces).

Tools require some special treatment. Tools are a class of artifact that operate on other artifacts, or on fixed objects. For example, a tin of beans might only be eaten if it is first opened with a tin opener. A locked door might only be opened by a certain key. A tool has all the attributes of an artifact, plus a list of the classes of objects on which it can operate, and optionally, a list of unique ids of specific objects on which it can operate. More general tools just list the classes of objects on which they operate, while specific tools will work only on specific objects. For example, a brass key might only open door id 5; a skeleton key might open any door. A hammer might work on any artifact, etc. The result of applying a tool to another object is stored either in the list of actions of the tool, or in the list of actions for the object. The tool action acts as a default (e.g. for a hammer: "hit X" results in "X breaks into millions of pieces") which can be over-ridden if the object itself has an action (e.g. for a bent key "hit with hammer" results in "the key is now straightened out").

Some thought must be given to the placement of tools in the game initialization process. For instance, some tools might be found only in the same room as the objects on which they operate. Furthermore, a key must not be located in a portion of the game that cannot be reached without it. A key should also not be located in the first room of the game, nor in the room that contains the door that it unlocks.

3.2.3.2 AM External Interface

The object manager will provide the following functions to be used by other CSCIs:

- An artifact namer: Given the unique id of a moveable object (creature or artifact), returns the short name of the thing.
- An artifact describer: Given the unique name of a moveable object (creature or artifact), returns a full description of that object
- An artifact initialiser: Creates an initial list of moveable objects (creatures and artifacts), and returns a list of their unique ids.
- An artifact placement checker: Given a list of artifacts in a room, checks that all placement constraints on those artifacts are obeyed.
- An artifact/action checker: Given the unique name of an artifact, and an action word, returns true or false depending on whether the action is valid for that artifact.
- An artifact/action updater: Given the unique name of an artifact, and an action word, updates the artifact's attributes. Returns true if the artifact was destroyed, false otherwise.
- An artifact metamorphasizer: Given the unique name of an artifact, and a description, changes the description of the artifact to the given description.

- An action lister: Returns a complete list of all action words used for all artifacts included in the game, including those action words that apply to whole classes of artifacts.
- A tool action checker: Given a tool and an action and an object, checks that the action is valid for that tool on that object.
- A tool action describer: Given a tool and an action and an object, returns a description of the result of using that tool on that object, and updates the tool and/or object description accordingly.

3.2.4 Creature and Fight Management CSCI (CFM)

CFM keeps track of all creatures in the game, including the player. It also manages fight sequences. Fights can be initiated by either a monster in the same room as the player, or by the player attacking a monster in the current room. If there are several monsters in a room, several of them can attack the player. They can also attack each other. Monsters do not attack one another unless the player is present in the room. As the player moves from room to room, monsters may decide to give chase. The decision about whether a monster should attack the player is calculated by generating a random number between 0 and 100, and comparing it with the monster's predilection. If the player attacks a monster, the monster attacked has its predilection changed to 100. When a player attacks a monster, the strength of the currently wielded weapon is deducted from the monster's strength. When a monster attacks another creature (monster or player) the strength of the monster's weapon is deducted from the other creature's strength.

If the monster attacks, one of its weapons is chosen at random. The strength of the weapon is deducted from the player's strength.

CFM also keeps track of the player, initializing and updating information about the player during the game, including the list of objects in the player's inventory.

3.2.4.1 CFM Internal Data

Creatures are represented using the following information:

- A unique id – This information is never divulged to the user. The player has id 0.
- A short name – used to refer to the creature when interacting with the user.
- A description – a detailed description of the creature. When the player enters a room, she will be given the short names of any monsters lurking there ("there is a dragon here)". If she looks at an object specifically ("look dragon") the detailed description will be given ("it is a small green dragon, with very fierce eyes, and dangerously sharp claws. There is smoke coming from its nostrils"). The detailed description may be empty, in which case the user will get a message such as "it is just a dragon".
- A list of weapons – these are the weapons used by the creature, represented as a verb, a short name, and a strength. The verb describes what the creature does with the weapon. The strength describes how much damage the creature does each time it successfully uses the weapon.
  E.g.
  Verb: *[scorches]*
  Short name: *[fiery breath]*
  Strength: *[20]*.
  The verb and the object are used to describe to the user what is happening ("The dragon scorches you with its fiery breath").
- Strength – An integer representing the remaining strength of the creature. When it gets to zero, the monster is dead. The player's creature always starts the game with strength 100.
- Predilection – A number between 0 and 100 indicating the percentage chance that the creature will attack without being provoked. 0 indicates the creature will never attack first, 100 indicates it will always attack first, any other value will be combined with a random number to find out whether the creature attacks for each turn that the player is in the same room as it. All monsters will always fight back if the player attacks them. Note that the player has a predilection too, but it is never used.
- Artifacts carried – A list of artifacts carried by the creature, represented using their unique ids. These will be available to the player once a monster is killed, and may include some of the weapons wielded by the monster.

Note that the player is represented as a creature too, and has all the attributes given above. The player starts with a strength of 100. Note also, the player can only wield one weapon at a time. The player has an additional attribute of score.

Monsters have one additional attribute:

- Motility – A number between 0 and 100 indicating the percentage chance that the creature will chase the player from room to room. After each move in the game, if the player has moved to another room, then each creature in that room that has been attacked is given the chance to chase. A random number is generated, and compared with the creature's motility, to determine whether the monster makes it into the next room.

3.2.4.2 CFM External Interface

- A monster angrifier – Given the unique id of a creature, changes it's predilection to 100.
- A creature strengthener – Given the unique id of a creature, and an integer (positive or negative), add the integer to the creature's strength. Returns true if the creature is still alive, false otherwise.
- A creature checker – Given the unique id of a creature, returns true if the creature is alive, false otherwise.
- A creature weapon lister – Given the unique id of a creature, lists the weapons wielded by that creature.
- A fight request – Given the unique id of a room containing the player, determines which creatures in the room will attack the player. Updates the monsters and the player's strength accordingly. Returns a textual description of the fight.
- A monster attacker – Given the unique id of a monster, calculates the result of the player hitting the monster with the currently wielded weapon, updates the state of the monster, and returns a description of the outcome.
- A monster chaser – Given the unique id of a two connected rooms, A and B, where the player is in B, determines whether any monsters in room A chase the player into room B.

3.3 Performance Requirements

The system should respond to each user input within 2 seconds. There are no other performance requirements.

3.4 Design Constraints

3.4.1 Standards Compliance

All language used in the game should comply with current University of Toronto guidelines for decency and equal opportunities. Usage of the game (other than planned testing episodes) on University of Toronto equipment shall be governed by the current guidelines for game playing.

3.5 Software System Attributes

3.5.1 Reliability

The system should never crash or hang, other than as the result of an operating system error.

3.5.2 Availability

There are no specific availability requirements

3.5.3 Security and Privacy

There are no specific security and privacy requirements, other than those generally governing use of student login accounts on University of Toronto computer equipment.

3.5.4 Maintainability

All code shall be fully documented. Each function shall be commented with pre- and post-conditions. All program files shall include comments concerning authorship and date of last change.

The code should be modular, to permit future modifications. Anticipated updates include changes to the sets of objects and their descriptions used during the game. These should be stored in a separate data file, rather than embedded in the program code.

3.5.5 Portability

The system should be portable to any UNIX system, including Linux. No other specific portability requirements have been identified.

3.5.6 Safety

The system should warn the use to take a break after every two hours continuous play to prevent eyestrain and repetitive strain injury. No other safety requirements have been identified.

3.5.7 Training-related Requirements

No specific training should be necessary for a user to begin playing the game. The game should give the user a brief guide for how to play (with some examples) when first invoked, before the first command prompt. Players should not be required to learn a list of commands before commencing play.

3.5.8 Packaging Requirements

The system should be packaged along with source code and all documentation, and be available for electronic transfer as a single compressed file to any purchaser. The uncompressed set of files should include a README file containing a minimal guidance for installing and running the game, including recompilation if needed. For recompilation, the system should include a makefile.

3.6 Other Requirements

There are no other requirements.

4 Appendices

There are no appendices

5 Index