RADAR

Risk Assessment and Decision Analysis Research Group

DCS 235 Software Engineering Group Project 2008-09

Problem Definition

Version 1.0, 18 Sept 2008

TABLE OF CONTENTS

1 Introduction

1. The aim of the 2008-09 software engineering group project (referred to as simply *the project* in the remainder of this document) is to create a stand-alone Java based implementation of a game based on the classic board game SNAKES AND LADDERS.
2. The rules of the game are described in the Appendix. You will need to read the rules carefully as there are subtle differences with the classic game.
3. The rulebook alone does not provide a sufficiently specific set of requirements for the project. Hence, the aim of this document is to partially clarify the requirements.
4. It is important to note that the special requirements of the project are such that there is no point in students attempting to cheat by downloading publicly available SNAKES AND LADDERS programmes from the internet. The special requirements force groups to design their own solid object oriented code that they fully understand.
5. We distinguish requirements as *must have, should have* and *could have* (Section 2) and list the requirements in Section 3. In Section 4 we describe what requirements need to be completed at each stage of the project. There are different requirements for BSc and MSc groups.
6. We have designed the requirements in such a way that you must apply solid object-oriented design principles in order to complete the project successfully.
7. You should read this document in conjunction with “Software Engineering Group Project 2008-09: Guidelines, Schedule and Assessment”.

2 General Structure for requirements

The functional requirements are broken down into three categories:

- **MUST HAVE (M):** These are the core requirements of the system. If you complete all these requirements perfectly (and no more), then the maximum score you can achieve for the final mark for the code is 65%.
- **SHOULD HAVE (S):** These are the additional requirements that improve the system beyond the M requirements. If you complete all these and the M requirements perfectly (and no more) then the maximum score you can achieve for the final mark for the code is 85%.
- **COULD HAVE (C):** These requirements are the gold plating of the project. Implementing all these (which includes innovative features at you own discretion) in addition to a complete set of M and S requirements will provide a maximum score of 100% for the final mark for the code.

As in any real project, you should not consider the set of requirements listed in this document as fixed. There may be additional requirements (as well as clarifications and changes to the current set of requirements) as the project progresses. These will be announced in lectures and on the course web site. There will certainly be one deliberate ‘new’ requirement (see requirement 7 in Section 3 below).

3 Requirements for BSc students

As discussed in section 2 we now describe respectively the *must have* (section 3.1), *should have* (section 3.2) and *could have* (section 3.3) requirements.

3.1 Must have requirements

1. **Command-based version of the basic game with no computer intelligence:** For this requirement you must provide a faithful implementation of the basic game as described in the Appendix. It is always assumed that there are two players: the user and the computer. The computer's moves are deterministic – it always moves forward the number of squares of the roll of the die. The user input and system output will be solely text based.

   Examples of commands could be:

   - "Start" (meaning start new game)
   - "Roll" (meaning roll the die)
   - "Move (X, n)" (meaning move player X to square number n)
   - "End" (meaning end the game)

   Examples of outputs could be:

   - "Roll of die was 6"
   - "Player X moved to square number n"
   - "Player X landed on a Snake at square number n"
   - "Player X wins"

   There must be no interaction with the system other than through the text window. However, to help players follow the game you must display the game board as a JPEG above the console window. We will supply the JPEG. Thus, your application will have the layout in Figure 1.

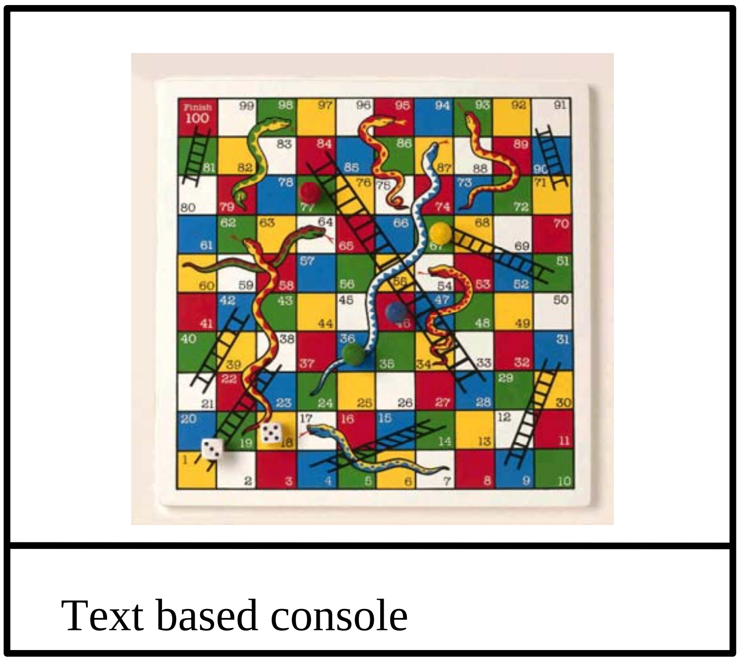

Figure 1: Console application layout

2. **GUI version of the basic game**: For this requirement you must implement a GUI interface to the game that enables users to experience the same 'look and feel' as the board game. This should include players having distinct 'tokens', which they physically move to the correct position after a roll of the die. The GUI version must be based around the same core classes and methods as the command-based version. Thus, for example, player X physically moving their token to square number *n* should invoke the programming equivalent of the command Move(X,n).
3. **Support alternative boards (GUI version only):** Although the board in the classic SNAKES AND LADDERS game is fixed (in terms of number of squares and the position and length of snakes and arrows), the same set of rules can be applied to any set of 'squares' and any configuration of snakes and ladders. This requirement is to produce a user-defined board. Specifically, the user defines how many squares in each direction, how many snakes and ladders, their length and positions. At start-up the user should be offered the alternative of playing with the classic board or defining their own.
4. **Implement undefined additional requirement**: To simulate real-world software projects there will be an 'announced' new requirement in week 16. Groups who have a solid object-oriented design should be able to easily accommodate the new requirement. Groups who do not have such a design (or who have plagiarized a solution) will find it extremely difficult to modify their code to accommodate this new requirement.

3.2 Should have requirements

5. **Save the game:** It must be possible to save the state of the game at any stage to a file. The format of this file is left to the application developers, ***but note that Java serialisation should not be used.***

6. **Load a game:** It must be possible to load a game previously saved to a file. The loaded file must restore the game to its state at the time it was saved.
7. **Add a scoring system**: With the basic board the scoring system is simply the running score of the number of times a player has won in the current session. With a user defined board, when the board is defined you must allow the user to:
   a) define the number of points to be awarded for winning a game
   b) define any number of squares as having a user defined value (which can be positive or negative). Any square with a defined value should have this value displayed on it. During play when a player lands on one of these squares, the value of the square is added to their score.
8. **On-line help system**: You should provide an html-based Help system that can be invoked from the GUI.

Could have requirements

9. **Intelligent computer:** The computer player can move forward or backward, just like the human player. For a number of reasons the optimal strategy on a particular move may be to move backward (for example, to 'bump' the opponent, to avoid a snake or to land on a high-value square). Your documentation will need to make clear exactly what strategy you have implemented and how.
10. **Innovative features**. Whereas requirement 9 makes up 10% of the marks the remaining 5% will be allocated to innovative features left to the Groups' discretion. This could cover anything from novel GUI features, through to features that improve the overall game experience. Where groups feel they have implemented innovative features they must make these clear in both the final report and PowerPoint slides.

3.3 Non-functional requirements

In addition to correct implementation of the functional requirements, your project code will be assessed against the following non-functional requirements:

- *Ease of installation*: This is the most critical non-functional requirement. You will have to learn how to 'package up' your code in such a way that it can be executed from a single file on any machine, without the need for any special installation instructions or additional files. Ideally this could be a jar, a bat or an exe file. Failure to meet this basic requirement will almost certainly result in failure of the whole project. In summary you must deliver your application in such a way that a 5-year-old child could install it and run it on any PC.
- *Portable*: Your application must run on any PC with an appropriate JVM installed on it.
- *Ease of use*: This is judged from the perspective of a SNAKES AND LADDERS player who has not yet played your version of the game. The elegance/attractiveness of the interface is also considered here.
- *Reliability/robustness*: The extent to which the code runs without failing. Some reasonable 'stress testing' must be performed.

- *Efficiency (speed):* Response time should be fast (both to GUI requests and computer player actions).
- *Efficiency (memory):* This will be judged on the amount of memory used when running, the size of the application files and the size of the saved game files.
- *Reusability:* This is the extent to which your application (or components in it) can be reused in other related applications. You will also have to explain in the final report how/what components could be reused in other applications.
- *Maintainability:* This is the extent to which it is easy to change your code in the light of new/changing requirements. A major test of this will be the ease with which you handle requirements 3 and 4.

4 Requirements for MSc students

Same as for the BSc students but with the following key exceptions:

- Must-have requirements are requirements 2 and 3.
- Should-have requirements are requirements 5 and 6.
- Could have requirements are requirements 4 and 10.

5 Incremental delivery (BSc)

In the lectures you will learn about the importance of incremental delivery. You should aim to provide regular increments (especially in term 2) that deliver increasing amounts of functionality. Your consultant will wish to see such increments. However, there are only two formally assessed increments that must be delivered:

1. Increment 1 (week 13): This must implement requirement 1. You can get the full allocated marks for a perfect implementation of this requirement. It makes sense to implement additional requirements if you are able to do so, but you will not be marked on these at this stage.
2. Increment 2 (week 22): This is the final code deliverable. You should implement as many of the requirements as possible.

6 Incremental delivery (MSc)

Although MSc students are encouraged to adopt the same incremental delivery approach described above for BSc students, they are only formally assessed on one code deliverable in week 22.

7 Appendix– The Rules

PLAYERS: Two

COMPONENTS: Board (as shown in Figure 2), one token for each player, a die

OBJECT: To be the first player to land on square 100.

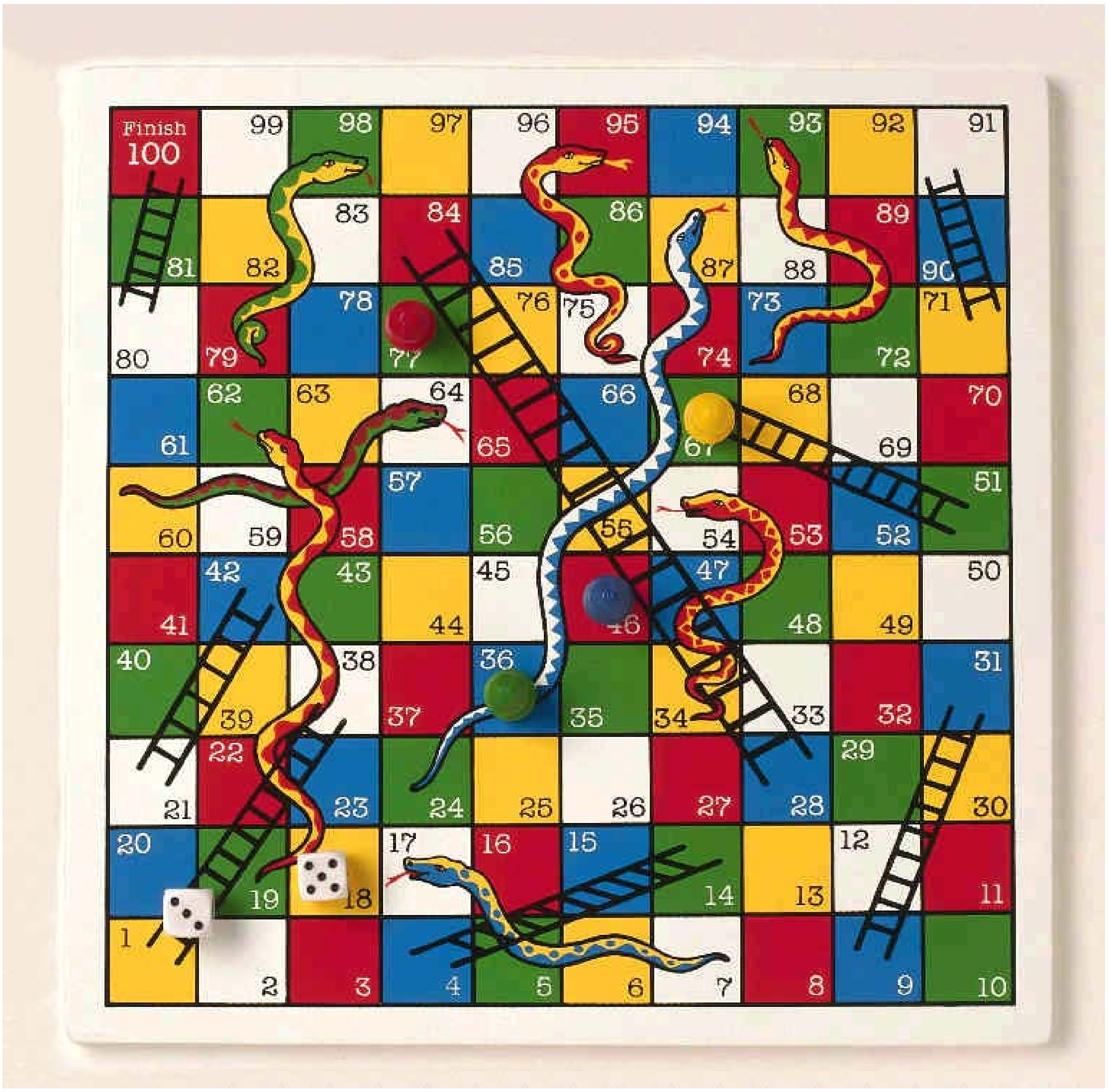

Figure 2 Standard Game Board

PLAY:

- To start, each player rolls the die to determine who goes first. The player rolling the highest number begins.
- Each player, in turn, throws the die. Apart from the very first throw (when they can only advance) a palyer can choose either to move forward their playing token the same number of squares as shown on the die or move back the same number of squares. The game begins at square 1 and finishes when a player reaches square 100.

- If a player rolls a 6 they get another go.
- Should a playing token land on a square bearing the bottom of a ladder, the playing token is carried up the ladder and placed in the square at the top of the ladder. If a playing token lands on a square occupied by the head end of a snake, it is carried down the snake's body to the square at the tail of the snake. All other parts of the snakes and ladders do not affect the course of the game.
- When a playing token lands on a square already occupied by an opponent, the opponent is 'bumped' and must start again from square 1.
- To land in the final square (100), a player must throw the exact number on the die. If the throw of the die turns up a number that is too high, the turn is missed.