Assignment 1: Board Game (X vs Y) 2019 Semester 1

15 MARKS

The aim of this assignment is for students to:

- **Analyse and document requirements and produce suitable design documents**
- **Develop an awareness of good object-oriented design principles (SOLID, GRASP, and MVC)**
- **Design and write maintainable code through an iterative process and refactoring**
- **Use a disciplined approach to team development through Design by Contract**
- **Work successfully in a team**

Overview
History is filled with conflict but here at RMIT we want to keep our lives trouble free and leave the conflict to the gaming arena!

Games and other media have many examples of "X" vs "Y" themes. Sometime goodies versus baddies and other times even baddies versus baddies. Examples include Aliens vs Predators, Vampires vs Zombies, Pirates vs Ninjas and even Orcs vs Elves vs Dwarfs vs Humans. But in this assignment we want you to use your imagination and come up with your own opposing forces while also learning good design principles, and gaining experience working on a team based software project.

Your mission is to develop a ***two dimensional, turn based board game*** *that is subject to a number of functional and design constraints listed below, but at the same time requires some creativity by formulating and documenting gameplay rules and design decisions.*

FUNCTIONAL REQUIREMENTS:

BOARD: Your game is to be based on the classic "board game" design where a two dimensional board is composed of a number of smaller play segments (such as squares on a chess board or real estate properties on a monopoly board). However, you are encouraged to use irregular geometric designs: curves, polygons etc. are all fair game, but for simplicity, the graphical representation will be a topographical (top down) view.

NOTE: In assignment part 2 the board will become more complex (blocked paths, obstacles etc.) so you may like to consider this in your design.

PIECES: Both opposing teams will be represented by a collection of pieces. At least three different categories of piece are required that move and behave differently (scout, commander, heavy, ranger, follower, healer etc.). For instance, a scout may be able to move further with each move, but can only sustain limited attack (since it has lighter armour), while a ranger can attack from further distance (and maybe only in certain directions), but can only move slowly or not at all and may have a cooldown after any action. You can have as many pieces on each team as you like (in any selectable combination) but less than 10-12 will keep things manageable. The behaviour of the pieces on each team are to be different (different capabilities, strengths, weaknesses).

TURN BASED: Play is to be turn based where each player makes a single move/action of a single piece at each alternating turn. A configurable time limit for each turn must also be provided.

GAME RULES: The gameplay is win/lose (as in chess) rather than score based and involves players capturing each other's pieces. We do not specify exactly how this occurs (this is left to your team to analyse and document). For example, it could be based on capturing a square, projectiles, or any other mechanism. However there must be some form of hierarchy or specific rules involved in the process of how a piece is defeated and captured ... think rock, scissors, paper (lizard, Spock) where scissors cut paper, rock blunts scissors, Spock disproves paper etc.

NOTE1: You cannot just do a basic game like chess or checkers with symmetrical team play where both sides have the same pieces.

NOTE2: You can make the gameplay either as friendly or combative as you choose, but avoid any graphic depictions that go beyond a reasonable PG/Family friendly nature (if in doubt your tutors and fellow students can offer guidance in the labs!)

DELIVERABLES AND MARKING BREAKDOWN:

Analysis and Gameplay 3 marks

Since this assignment requires some analysis and synthesis of game behaviour you need to document the gameplay in a suitable fashion. You can use a combination of use case, sequence, and activity diagrams (whichever you feel best expresses your gameplay) as well as written descriptions. You must provide at least 2 formal diagrams which capture all gameplay/rules as well as additional written descriptions.

System Design 3 marks

You should provide at least 2 class diagrams that show your class structure in terms of a Model View Controller Approach (MVC).

You should provide at least one non trivial sequence diagram which illustrates an example of gameplay (beyond that which is to be implemented according to the section below).

System Implementation (Prototype) 6 marks

Your assignment will be extended to a full game for Assignment 2 but for Assignment 1 you should create a working prototype which shows the game board, the pieces and allows players to choose and move a piece from each opposing team. At this stage you do not need to implement the capturing phase.

Your implementation should be in Java using the standard libraries of Java SE such as AWT/Swing and the Java 2D API. Note that since the focus of this course is on OO design you should not dedicate excessive effort to the graphical representation and can use any freely available media assets within their licensed terms of use.

Non Functional Requirements 3 marks

- Code should follow the SOLID and GRASP principles whenever possible
- Evidence of design by contract (class invariants, pre and post-conditions for methods)
- Separation of Model, View and Controller

Appendix A: Code Quality Guidelines

Avoid:

- ☐ Redundant/obvious comments that can be inferred from the source code
- ☐ Commented out code
- ☐ Dead functions
- ☐ Large classes and methods
- ☐ Data only classes
- ☐ Incorrect class hierarchy
- ☐ Duplicated code in classes and methods
- ☐ Feature envy (see lecture on refactoring)
- ☐ Inappropriate intimacy (see lecture on refactoring)
- ☐ Vertical separation (see lecture on refactoring)
- ☐ Switch statements (use polymorphism wherever possible)
- ☐ Long parameter list in methods and interfaces (use encapsulation)

Use / Allow

- ☐ Readable and maintainable code
- ☐ Meaningful and consistent naming convention
- ☐ Low Coupling between classes
- ☐ High Cohesion in classes
- ☐ Consistent indentation and spacing
- ☐ Comments that add value above the source code e.g. explaining algorithms, dependencies or reasons for design choices
- ☐ Javadoc for core interfaces/classes, especially where there is a boundary between implementation from different team members