HexAround: A Strategy Game for CS4233

Rules & Developer's Guide

Version: 1.1

Gary Pollice

February 1, 2024

Abstract

HexAround is a game of strategy. The goal is to encircle your opponent's primary piece. Two players place and move pieces, usually named after animals or insects, on a two-dimensional board that is tiled with hexagons. The game is highly configurable in its ability to define different types of pieces, how they might move, and additional rules. Students in CS4233 will create a game manager that takes input from players and contains the logic to manage the game state, and determine the winner.

This manual contains the rules of the game and helpful hints for developers who will produce software implementations of the game for CS4233: Object-Oriented Analysis and Design.

Contents

1 Introduction

HexAround is a game of strategy that is played by two players on a two-dimension board that is tessellated with hexagons.

The game is simple. There are two players, Red and Blue. The goal of the game is to surround your opponent's primary piece (usually called the Butterfly). While the game is played on a board consisting of hexagons, the board has no specific dimensions. The active board grows and shrinks as the game progresses. The current size of the board is a function of the number of pieces in the particular version of the game and how they are currently configured on the board.

Each variation has one or more types of pieces specified for the game. Each type of piece has its own characteristics that describe how it can move and its "powers."

2 HexAround rules

HexAround is a strategy game for two players. Each player controls a group of creatures with the objective of trapping the opponent's Butterfly first. A Butterfly is trapped when it is completely surrounded on all sides by other creatures, regardless of their owner. (Figure 1)

If both Butterflies are surrounded by the same move, then the game results in a draw. If a player is not able to move or put new creatures on the board, his opponent is declared winner.

2.1 Playing the game

You can do only one of two possible actions during your turn: add a new creature or move a creature already on the board.

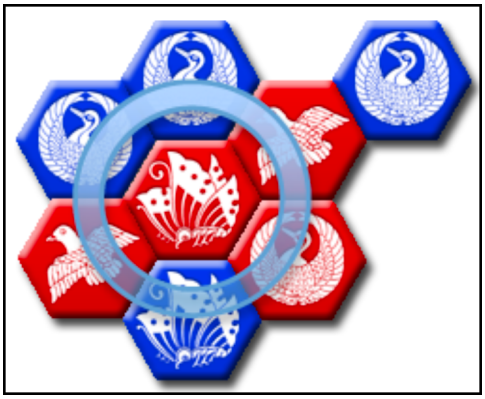

Figure 1: Red Butterfly is surrounded; Blue wins.

Add a new creature

The Blue player makes the first move and places a creature from the reserve on the board. Red moves next, and **must** place a creature next to the piece that Blue placed.

After each player places their first piece additional creatures must be added next to a creature of the placing player's color. However, a creature cannot be deployed into a position next to an enemy creature, even when that position is also next to a piece of the placing player. (Figure 2)

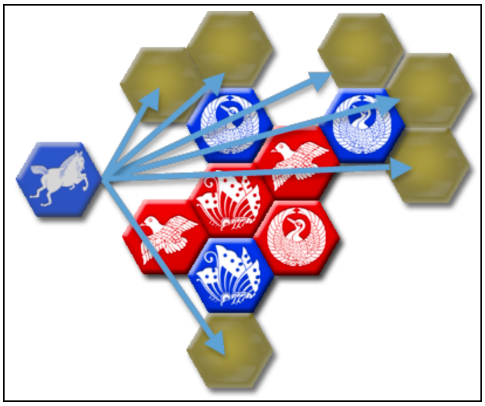

Figure 2: Valid placement locations.

Moving a creature already on the board

Creatures already in play can be moved to a different position. Every creature has its particular way of moving. The way that every creature moves will be described later. However, for any movement certain constraints must be obeyed. These are:

- **Connectedness:** At all times the creatures on the board (the colony) must form a connected collection of creatures. Every creature, except for the first one placed on the board must have at least one other creature directly adjacent to it in one of the six possible neighboring hexes. Figure 3 illustrates this. The butterfly cannot move to the hex directly below it because it would disconnect the colony.

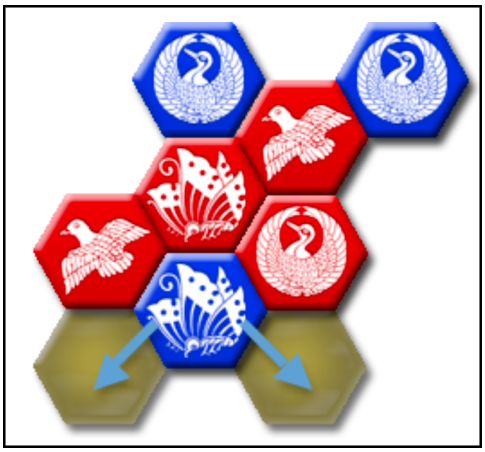

Figure 3: Connectedness. Blue Butterfly has only two possible moves.

  It is important to remember that at all times every creature should be next to one or more creatures from any player, forming one connected colony. It is not legal to move a creature in such a way that would split the colony in two or leave any creature disconnected from the rest. This is a rule that can be used to both players' advantage, since it can immobilize an enemy creature by simply moving a creature to a position where it is only connected to that enemy creature. See Figure 4.

  For creatures that do not fly, they must move in a path so that at each hex they pass through, this connectedness property must apply.

- **Dragability:** All Walking and Running creatures must be able to be

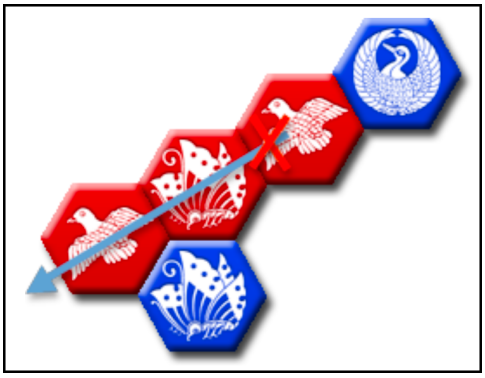

Figure 4: Connectedness. Red Dove cannot move.

"physically" dragged in a horizontal way from their starting positions to their end positions. For example, if five creatures are surrounding a Walking creature, it cannot move. Figure 5 illustrates this.

Figure 5: Dragability. Red Butterfly cannot move.

One way to think about this is that the creature takes up the full area of the hexagon on which it rests. Therefore, there must be at least one empty hexagon next to the target hexagon in order to slide the piece to the target.

2.2 Creature categories and attributes

HexAround is a configurable game. This means that there are different types of creatures and each type of creature belongs to a category that has general

characteristics. These general characteristics determine the ways that the piece type may move, for example a Walking creature walks one hex at a time, but has an upper limit of how many hexes it may travel in a single move. In a particular game variation other attributes, such as how many hexes a specific piece type may move in a given turn, are added to the general category's characteristics. Every piece must have exactly one of Walking, Running, Flying, or Jumping. Piece types may belong (or have) more than one category. Table 1 in Section 2.2 shows the possible combinations, unless prohibited by other constraints.

Butterfly

The Butterfly is a special Walking creature that moves one hex at a time. Each player has only one per game, and if it gets surrounded, you lose. In any variation, the Butterfly must be put on the board within the first four turns (one turn is when both players have made a move). And if it has not been entered by the fourth turn (or it has been removed due to a Kamikaze attack), the only valid move for the player is to place the Butterfly. If the Butterfly cannot be placed, then the opponent wins. Butterfly has an additional property called "QUEEN." This is simply to identify it as the unique piece that, when surrounded, ends the game. This property is not shown in Table 1. The Butterfly can only have WALKING and QUEEN properties.

Walking

A creature that walks will move one hex at a time, as if it were sliding, up to its maximum number of moves declared in the configuration. If a piece type is able to walk $n \geq 1$ steps, this should be thought of n individual walks of one step each. At any step of a walk, the piece may not be disconnected from the colony. Walking creatures may only move to empty hexes. Figure 3 illustrates the Butterfly walking.

Running

Running creatures **must move the exact maximum number of hexes** defined in the configuration. They may not move to the same hex twice in one turn. For example, a Running piece with a maximum that moves two hexes cannot move to an adjacent open hex (assuming that it does not violate any other constraint), and then back to the starting hex. As with

Walking, $n \geq 1$ steps, this should be thought of n individual walks of one step.

Figure 6: Running.

Flying

A Flying creature moves up to its maximum number of hexes ignoring other creatures. It should land in an empty place unless other abilities (like Trapping) allow it to land in an occupied place. Figure 7 illustrates flying. Flying creatures may not move if they are surrounded.

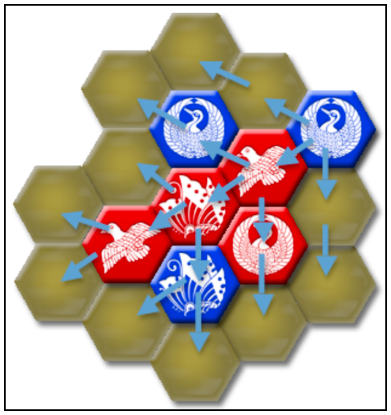

Figure 7: Flying.

Jumping

Jumping is like Flying except that the the piece **must** move in a straight line. A benefit of Jumping creatures is that they may move even if they are surrounded—as long as the colony stays connected.

Figure 8: Jumping Dove.

Intruding

Intruding creatures may move through occupied places. The movement is similar to Walking creatures (i.e., one hex at a time on the path to the

destination.) The main difference is that it can walk over any other pieces already on the board. An Intruding creature must have an extra ability, for example Trapping, in order to be useful. **Trapping is the default ability for Intruding creatures if none of the others is defined in the configuration.** In Figure 9 the Red Sparrow intrudes over the Red Dove. If the Sparrow has the Trapping (default Intruding ability) ability, then the Dove cannot move; otherwise the Dove can move.

When placed on the board, the Intruding creature is placed in the same way as any other character. It cannot be placed on top of another piece.

Note: When there are two or more pieces on a single hex, forming a stack, the "color" of the stack is the color of the top piece.

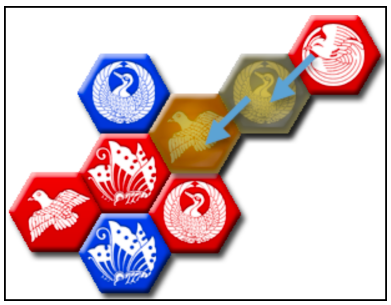

Figure 9: Intruding Sparrow intrudes on the Dove.

To summarize what an Intruding creature can do:

- Intruding creatures may move over other creatures or stacks, but they can also move on empty hexes as long as the colony remains contiguous (like Walking creatures).
- Intruding creatures may end up on top of a stack, unless other conditions prevent it.

Trapping

A creature that traps can move to a space already occupied by any other creature to immobilize it there. The creature below the trapper will not be able to move until the trapper moves to another space. This means that it is possible to have a stack of pieces on a hex. The Trapping piece must be the top piece and all pieces under it are trapped. If it is below the top of the stack, then just the pieces below it are trapped until the Trapping piece moves.

Figure 10: Blue Crab traps Red Crane.

Swapping

A creature that swaps exchanges its original place with the creature on the destination hex. *Swapping creatures should be able to physically get to their destination based upon other constraints.*

Swapping pieces cannot swap their place with a Butterfly. Also, if the piece is trapped, it may not move at all, using the Swapping property.

Kamikaze

A creature that has Kamikaze ability is one that, allows a piece to attack another piece, as if it had Swapping ability. However, the attacking piece **will be destroyed** and the victim is returned to its owner's hand for future placement. Kamikaze attacks can also be performed against your own creatures, so they can even save a Butterfly in distress! If a Butterfly is destroyed in a Kamikaze attach, it **must** be returned to the board on the next move for that player. The only exception would be if the fourth turn has not yet been reached; in that case, normal rules apply for that Butterfly.

Hatching

Hatching abilities allow creatures of the same player to be added in a hex adjacent to them, even that hex is also adjacent to an opponent's creature. In Figure 11, the Hatching Blue Crane allows Blue creatures to be added to the two hexes with the blue circles on them.

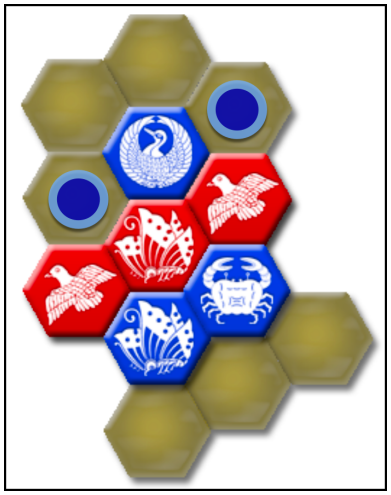

Figure 11: Hatching.

Valid attribute and category relationships

Section 2.2 has a lot of categories and attributes. It may be difficult to determine which can be used in combination with others. Table 1 is a single source of reference for finding these out. Butterfly is not included as a category. Only one Butterfly is allowed for a player and its complete capabilities are described in Section 2.2. Remember, these are *possible* pairwise relationships, not necessarily ones that would be used in a particular configuration.

	Walking	Running	Flying	Jumping	Intruding	Trapping	Swapping	Kamikaze	Hatching
Walking					✓	✓	✓	✓	✓
Running					✓	✓	✓	✓	✓
Flying					✓	✓	✓	✓	✓
Jumping					✓	✓	✓	✓	✓
Intruding	✓	✓	✓	✓		✓	✓	✓	✓
Trapping	✓	✓	✓	✓	✓			✓	✓
Swapping	✓	✓	✓	✓	✓			✓	✓
Kamikaze	✓	✓	✓	✓	✓	✓	✓		✓
Hatching	✓	✓	✓	✓	✓	✓	✓	✓	

Table 1: Relationships pairwise combination.

3 Developer notes

This section contains information that should be helpful for students who implement a game manager for a course project.

3.1 The board

The HexAround board is unbounded. When the Blue player makes the first move by placing a piece on the board, that hexagon is the "origin hexagon." Hexagons are identified by a pair of integers. The origin hexagon is at location (0, 0). It provides a point of reference for all other hexagons. Figure 12 shows this and illustrates how hexagons are identified and related to each other. We use a Euclidean-style (x, y) labeling to identify the hexagons.

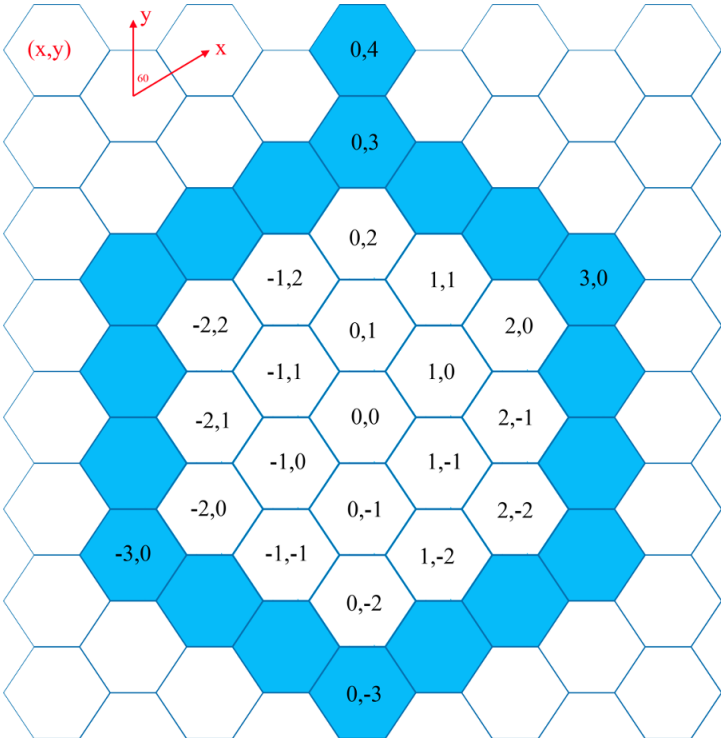

Figure 12: HexAround board.

4 The starting code

The starting code is fairly simple. There are just two significant interfaces or classes and a few enumerations and records. Figure 13 is a UML diagram showing some of the relationships between them.

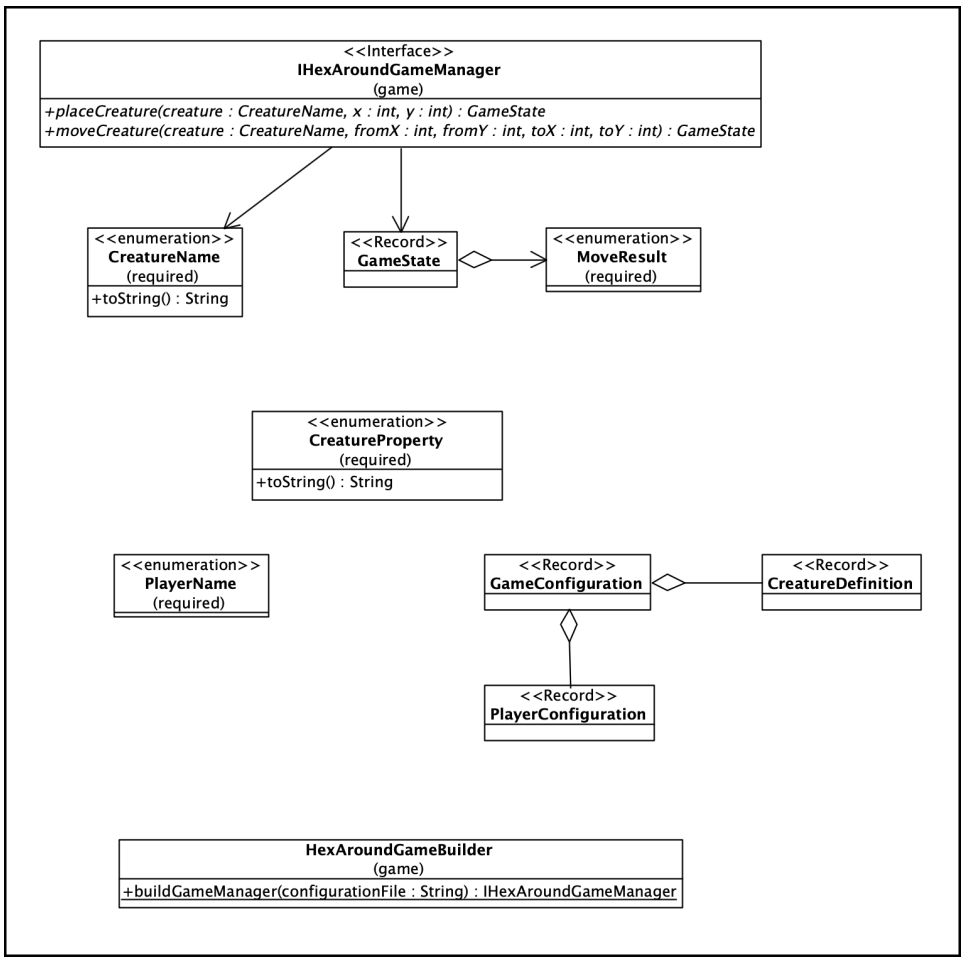

Figure 13: HexAround starting code.

You will have access to one or more videos that review the starting code. You will modify the `HexAroundGameBuilder` to create an instance of a game manager using the `GameConfiguration` record. **You may not change anything in the `config` package, or the `grammar` directory**. The game builder should create the implementation object of the game manager. The

game manager should not do all of the work in its constructor. Apply the Builder pattern (which will be covered in the class).

NOTE: In all of the files in the starting code, you are expected to read the comments, especially about what you may and may not move or modify.

5 Assignment

Make sure that you read the complete assignment specification.

This assignment is the culmination of the term. If you had success in the previous assignments, you should be in good shape for this final one. There are three levels of tests that we will run for this assignment submission. Each one adds points to the final number of points for the correctness section. The levels are described later in this assignment specification. **Each level is specified by the features of the game that you must implement**. In addition, there will be one or more submissions before the final that will be graded as programming assignments. These are designed to test just some basic capabilities and ensure that you are not putting off the project until the last minute.

NOTE: You may only use the libraries that are included in the starting code. If you think you need more, get approval first from the instructor.

5.1 General requirements

Your code will be reviewed for readability, intentionality, organization, and application of principles and patterns. We will also run your tests for code coverage. You should have at least 93% code coverage (branch and line) on code you write. You will also maintain the TODO.md indicating your development using TDD.

Organization is about how you have grouped classes and packages to show the organization of your design, i.e., where you think the major components of the game manager are. For example, you might have a package named `rules` where you have all classes that are used to apply the rules and check to see if they have been violated. *Your package design should reflect your project's design structure.*

You are expected to write code that follows these guidelines:

- Has consistent formatting and style.
- All public methods should have Javadoc comments. You do not need them on private methods, but the methods should be intentional. If there is doubt, add method comments.

- Minimize inline comments (//). If you name your methods and variables well, and write clear, intentional code you should not need these. This is not to say that all inline comments are bad. If there is something about an algorithm that is not clear, such a comment may be appropriate, but comments like the following are useless and make the code less clear and more difficult to read and understand.

```
//if the player is not in the players list
if(!players.contains(guess.playerName())) {
  ...
}
//checks if only one player remains
if(playersLeft.size() == 1) {
  ,,,
}
```

- You only expose what is necessary. This is especially true for public fields, unless they are in a class that the client would never access; and even then, be very careful about it.

- If you extract code from another author, make sure that you cite it in your code where you use it and also in the last part of the TODO.md.

The code should be your own, almost exclusively, unless you extract a small snippet from a source. If you write code for an algorithm that you searched for, cite the page or book that you found it in your code as a comment. If you use a generative AI that writes, or produces example code, copy the whole session, or a shared link to the session into a separate file called Sources.md.

If you use design patterns, identify them, where they are applied, and why you chose them. Put this information at the end of your TODO.md.

5.2 Level 1 requirements

Implement a game that can be played (i.e. moves made by the client, which is my tests) that has the following characteristics and features:

- Support for pieces with the following properties: Butterfly (QUEEN), Walking, and Flying.

- Placing pieces on the board in legal positions. That is, they are placed according to the rules on adding a new creature in Section 2.1.

- Moving pieces, but you do not have to check for connectedness at each step of the move. You also do not have to check to see if a piece can "slide" on the path. See Figure 5.
- You can assume that all moves given by the client are legal for this version. You do not have to check for any invalid move.
- Determine if the game is won by a player. You do not have to check for a drawn game.
- You do not have to check to see if the Butterfly has been placed by the end of the fourth turn.
- Keep track of which player is moving.
- You must determine that the colony is a single connected colony at the end of each move.

5.3 Level 2 requirements

In addition to all of the Level 1 requirements , you must also implement the following for Level 2.

- You must check for the Butterfly placed on the board by the end of the fourth turn.
- Add Jumping pieces. However the Jumping piece must land on an unoccupied hex.
- Ensure that pieces can be dragged. See Figure 5.
- You must handle situations where a move is invalid for this version of the game. When that occurs, you will return a game status with a `moveResult` of `MOVE_ERROR` and provide a message that explains the problem. If an invalid move is received, the next move received is for the same player that made an error.
- Implement the Kamikaze attribute.

5.4 Level 3 requirements

In addition to the Levels 1 and 2 requirements the following will be implemented.

- Implement the check for continual connectedness of the colony at each step of a move.
- Add Running creatures.
- Add the Swapping attribute.
- Allow stacks of up to two creatures. If a creature with the Intruding attribute ends up on an existing stack of two, it is an invalid move.
- Implement Intruding. However, there is no default Trapping attribute. It is possible for an intruding creature to have no additional attributes.
- Check for a drawn game (when a move causes both Butterflies to be surrounded).
- Check for valid placement on the board according to the description of adding a new creature to the board in Section 2.1.

6 Submission

Submit the complete project as an IntelliJ IDE project. Make sure to include the libraries. Your code should run with the libraries that were provided to you with the starting code. We are using JUnit5 testing framework. Do not use Windows specific path names. All paths to your test configuration files should be relative to your project and start with "configurations/" followed by the path to the specific configuration file in the configurations directory.

The submission should be zipped archive that includes the libraries in the IDE project. **Set the Java language level to 20 or below**.