Tiger Zone

Software Design Specification

by Team I
Steven Remington
Aubree Richards
Kyle Hardin
Logan Barkes
Kenan Yildirim
Michael Lowe

Contents

1. Introduction

Tiger Zone is a software based game meant for two players which function around gaining control over territory through the placement of tiles and follower entities. A computer AI will be implemented to play against a user and other AI's. This game will be developed through the Windows operating system in the Java programming language.

The final program should meet the following generic requirements:

- The application should be able to provide a playable experience following the basic guidelines of a two player Carcassonne game outlined by Rio Grande's official rules which will be enumerated later.
- It should be able to have an adequately powerful artificial intelligence engine made to play the game to the best of its ability, being able to play 35 moves in less than two minutes.

These requirements may be adapted as the progress moves forward. This document should serve as a guideline for development for our group and description of our goals to Professor Dave Small.

2. Design Considerations

This section details the aspects of design the will help us arrive at the early core decisions of the development of the Tiger Zone application.

2.1. Design Artifacts

To aid in the design of progress tracking of the application, our development will be split into weekly sprints designed to accurately track and motivate the completion of Tiger Zone application in a timely manner. Individual sprint goals function to give an accurate overview of each week in the application development. A product backlog is utilized to manage the overall tasking left for completion of the application. Our product backlog is hosted through easybacklog.com. The Sprint burndown chart is used to give weekly tasks with time estimations allotted to each task.

2.1.1 Sprint Goals

There will be in total four sprints to cover the four weeks that this project encompasses And a respective goal for each sprint to describe the expected progression of the project. These are tentative sprint goals with the intention of showing expected progress of the project.

Sprint 1: Familiarize the group with the domain in order to create a functionally efficient design through a hexagonal architecture and begin to implement testing modules.

Sprint 2: Have a functional Board with Tile Manager, populated with tile objects, with the ability to validate or reject possible tile placement on the board.

Sprint 3: Fully implement scoring and artificial intelligence to allow for a computer game to be played against itself.

Sprint 4: Finish implementation of network communication and any required translation from standardized data.

2.2. Assumptions and Dependencies

- Cross Compatible platform to support multiple operating systems
- Two player limit
- Must be network compatible

2.3. General Constraints

This section enumerated the limitations and constrains our group has arrived upon when designing Tiger Zone.

- Standards compliance: our AI must strictly conform to the game's rule in order to avoid being disqualified
- Interface/protocol requirements: our program must be able to accept and output tile placements
- Performance requirements: our AI must be able to decide on a tile placement within a sufficient time limit in order to avoid disqualification
- Network communications: our program must be able to communicate with the external tournament server via TCP.
- Verification and validation requirements (testing)

2.4. Goals and Guidelines

The goals for this project is to create an artificial intelligence to solve the Tiger Zone board through competition with itself or other artificial intelligence objects.

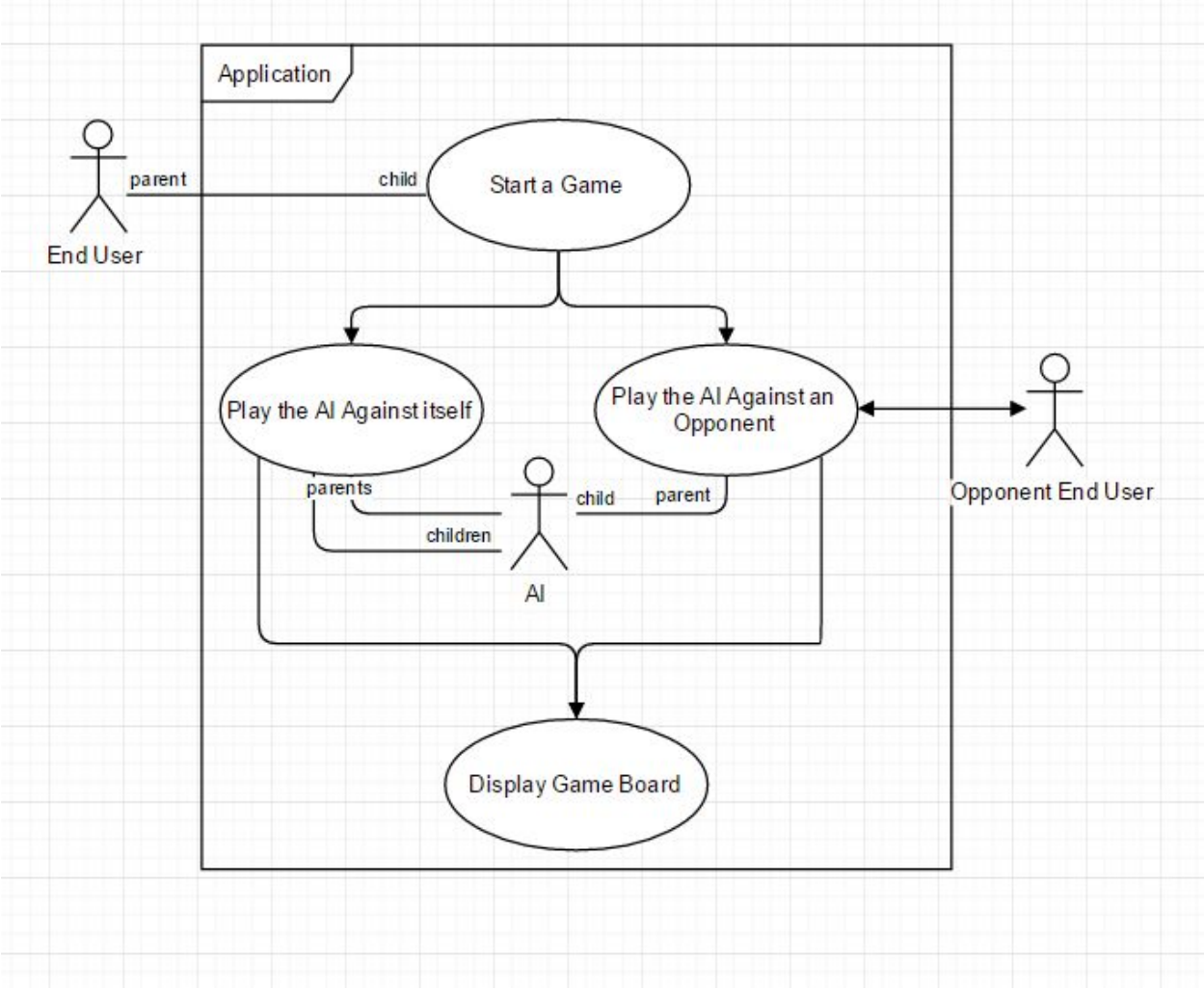
- Focus on speed of move decisions over memory efficiency
- A strict, correct, and consistent Implementation of the game's rules takes precedence over aesthetics of the UI
- Design to maintain the behavior of the Original Domain, based on Rio Grande's Carcassonne, until otherwise adapted
- The AI will form a decision tree containing the possible tile placements and utilize heuristics in order to determine which branches result in minimizing the opponent's score (minimax algorithm)

2.5. Development Methods

We will be utilizing an agile approach using Domain Driven Design. One week sprints will be implemented to check progress and ensure that our group is moving forward. A Product Backlog and a series of burndown charts are used further manage progress and tasking within the group. The product backlog is hosted through easybacklog.com, whilst the burndown charts are tracked via Google Docs. Diagrams to illustrate overall flow of the program will be created using draw.io.

2.6. Use Cases

- Have the Tiger Zone AI play against itself
- Have the Carcassonne AI play against other entities (i.e. a human, or another AI)
- Display Current game board



3. Architectural Strategies

- Programming Language: **Java**
    - Portability was required due to development across multiple operating systems
- Design Strategy: **Hexagonal Architecture**
    - This strategy is used due to changing requirements

- System input used to navigate and start games
- System/Network input used to drive enemy game movements
- Error Logging to be done in a file export done upon completion of execution

4. System Architecture

OVERVIEW:

The Tiger Zone System Architecture is defined through four primary modules: Board/Tiles, Rules/Scoring, Artificial Intelligence, and Input/Output. This application will be developed by the members of Team I and as such we will not be repurposing off the shelf items for integration into the system.

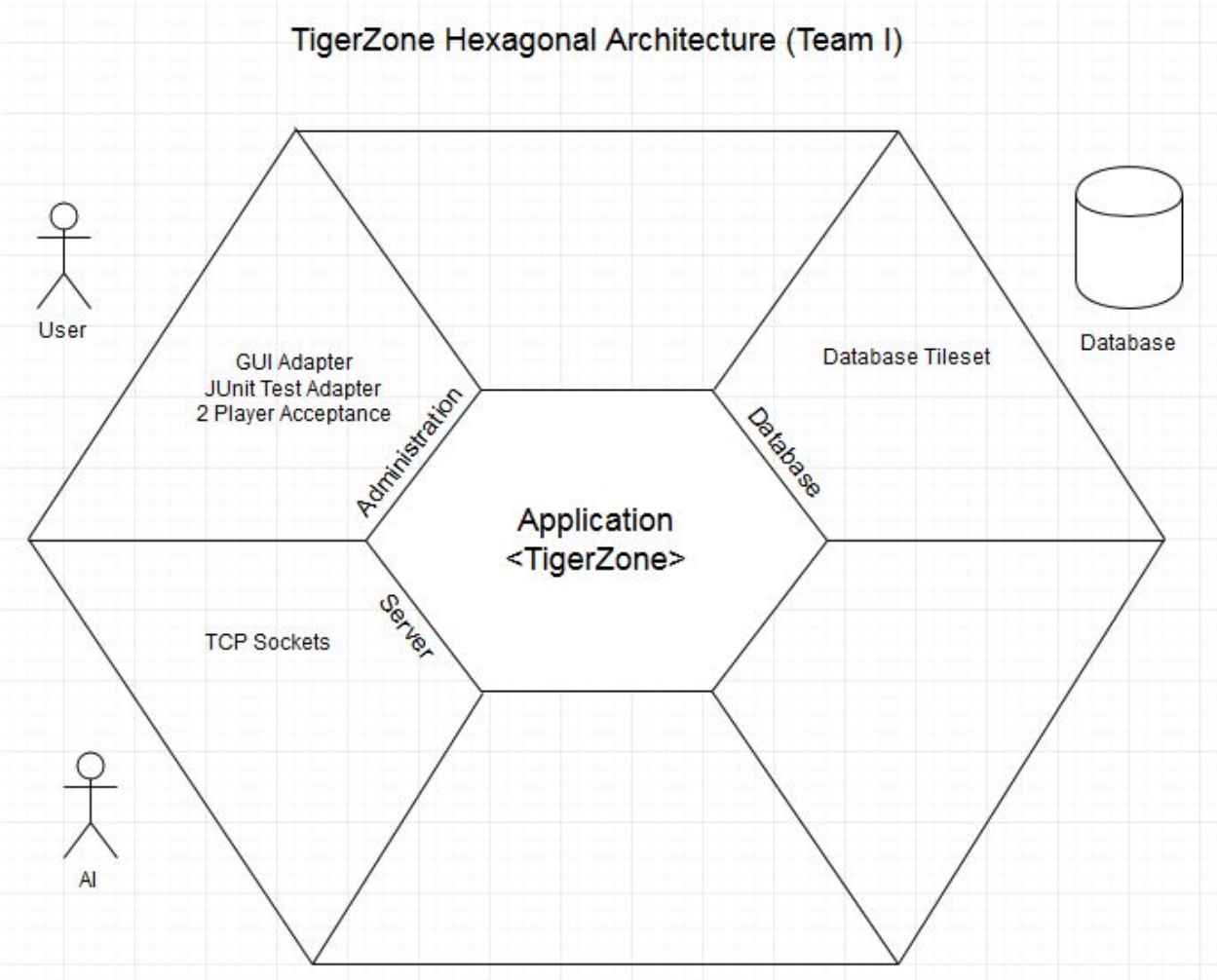
The Board module's primary functionality is to hold the current board state, including the placement of new tiles on the local state. Subsystems of the Board module include a Board state and a tile manager.

The Rules Module is meant to validate the tile placements of both players, as well as manage the runtime and end game scoring. THis module will be closely tied with the Artificial intelligence module due to the requirement of the AI making valid decisions for tile placement

The Artificial Intelligence module is used to play against itself and other artificial intelligences by interfacing with the Board and Rules modules through the input/output module. This module may be broken into two primary modes of thought: placement of tiles and placement of followers.

The final module deals with Input and output, including GUI and Network interface. Input and Output will also handle any necessary translation that would stem from conflicting data representation as well as an anticorruption layer to make the overall software more robust.

The driving architecture for TigerZone is the **Hexagonal architecture**. At the center is the TigerZone application, bounded by two ports: The database and administration. The stack of tiles received from the server will be treated as the database system. The user interface, test adapter (JUnit), and the two-player acceptance will be the adapters that communicate with the administration port.



4.1. Board Subsystem Architecture

The board houses a mapped **gameGrid<Position,Tile>,** as well as acts as a facilitator to keep both placed tiles and a stack of tiles not yet placed. The purpose behind this is to effectively mitigate the responsibility each instance of the object **Board**; which in turn allows for the encapsulation of the parallel architectures to concisely communicate with each other in this Board subsystem.

List of interactors:

- Tile
- Rule
- gameGrid<K,V>
- RuleEngine

4.2. Rules Subsystem Architecture

The subsystem is based off a modified creational pattern, Factory Method; where a specific rule is generated from the object **RuleEngine.** This allows for the **RuleEngine** to act as a factory in order to create different implementations of the interface **Rule** as needed by the overall Rules Subsystem. This design technique is applied to all objects that interact in running the overall System.

List of interactors:

- Rule
- RuleEngine
- SideMatchRule
- TigerJungleRule
- TIgerLakeRule
- TigerTrailRule
- PlacementRule
- AdjacencyRule

4.3. Artificial Intelligence Subsystem Architecture

The Artificial Intelligence(AI) Subsystem acts as an abstraction of possible AI extensions and addons (robust solution). This level of abstractions allows for creation of an AI player, as well as how to grab the instance ID of the player.

List of interactors:

- AIPlayer

4.4. System Protocol Architecture

System protocol acts as the buffer zone between all other subsystems and the I/O(input/output) subsystem. This protocol allows the communication of all instances of all subsystem components. This will act as a control flow for the overall play of Tiger Zone as well as serve as a structural timeline of events during the Runtime.

4.5. Input Output Subsystem Architecture

Input and Output will handle any necessary translation that would stem from conflicting data representation as well as an anticorruption layer to make the overall software more robust. The Input architecture will receive commands from an user, an AI, or a network depending on which ports will be used (Testing/Playing). The output architecture will take the commands the AI would like to complete and translate those commands to the network, UI, or opponent AI. These two architectures will make up the Input/Output subsystem architecture.

5. Implementation Review

Preface:

Over the development life cycle of this project, there were four, week long sprints with the intent of delivering a functioning implementation of Tiger Zone by the project deadline. Between the first two weeks, System design considerations and initialized constraints were created because of the customer requirements. Following that process a prototype of the board was developed. The following two weeks were dedicated towards system architecture updates, refactoring, and adjusting to user requirements.

Below is a brief timeline of the team's tasks **weighted in hours**.

Sprint 1
Involved initial setup, including assigning roles, creating a backlog with initial goals, and selecting an architecture, testing framework, and development platform. It was determined that development would be done in Java using a hexagonal architecture, whereas tests would be executed using JUnit. This testing framework integrates well with Java. All goals were met at the end of this sprint.

SPRINT 1 BURNDOWN							
	Day						
Item(weight)	Monday 10/31	Tuesday 11/1	Wednesday 11/2	Thursday 11/3	Friday 11/4	Saturday 11/5	Sunday 11/6
System Platform Collaboration	4	4	2	0	0	0	0
Create Entire Backlog	14	10	10	2	0	0	0
Write Initial Design Documentation	10	9	9	6	4	2	0
Setup Testing Framework (JUnit)	3	3	2	1	0	0	0
Assign Roles	2	1	1	0	0	0	0
Define Initial Classes/Modules	8	6	5	2	2	1	0
Lay Out Architecture	16	16	16	12	8	5	0

Sprint 2

This sprint contained fewer goals with higher time requirements. The main goals were completing a basic UI that can make a simple tile placement as well as implementing tile placement rules. Complete rule implementation is a large undertaking and was not completed by the end of the sprint. There are several fringe cases that the rules must cover, and these are taken care of in the next sprint.

SPRINT 2 BURNDOWN							
	Day						
Item(weight)	Monday 11/7	Tuesday 11/8	Wednesday 11/9	Thursday 11/10	Friday 11/11	Saturday 11/12	Sunday 11/13
Develop Tile/Board Classes	32	28	24	20	18	14	10
Tile Placement Verification	40	40	38	34	34	34	0
Create Basic UI for Testing	34	28	24	18	14	8	0
Begin Implementing Rules	40	36	32	26	22	22	20

Sprint 3

This sprint is a continuation of the rules, AI, and lake/game-trail completion. Completing the rules for matching sides and checking adjacency was simple, but checking for completion of lakes and game-trails required more time and leaked into the next sprint. By the end of this sprint, the UI could play a full game (minus scoring) and the AI was in the process of playing a full game.

SPRINT 3 BURNDOWN							
	Day						
Item(weight)	Monday 11/14	Tuesday 11/15	Wednesday 11/16	Thursday 11/17	Friday 11/18	Saturday 11/19	Sunday 11/20
Develop Tile/Board classes	8	4	0	0	0	0	0
Allow AI To Make a Basic Move	28	24	21	15	10	6	0

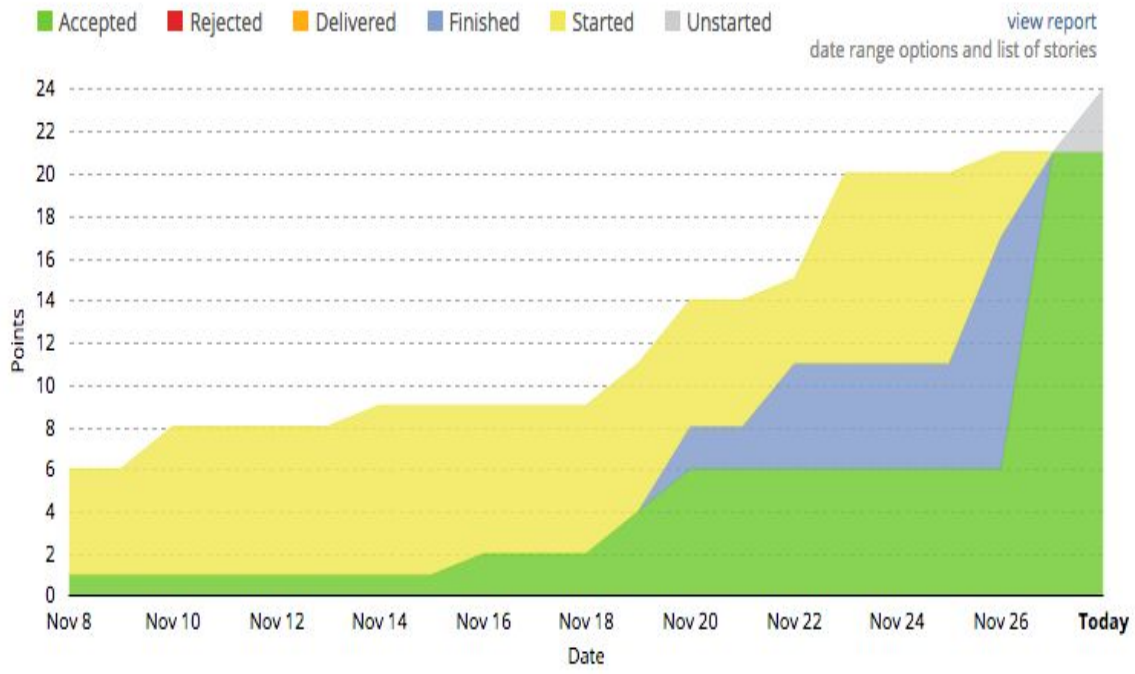
Make Side/Adjacency Rules	6	3	0	0	0	0	0
Begin Road/Lake Completion Checks	26	24	20	18	18	16	13
Create Basic Mock Server	20	20	14	10	7	4	0
Implement Scoring on Completion	25	23	20	16	16	15	15
Keep Adding Rules for Edge Cases	20	16	12	12	6	0	0

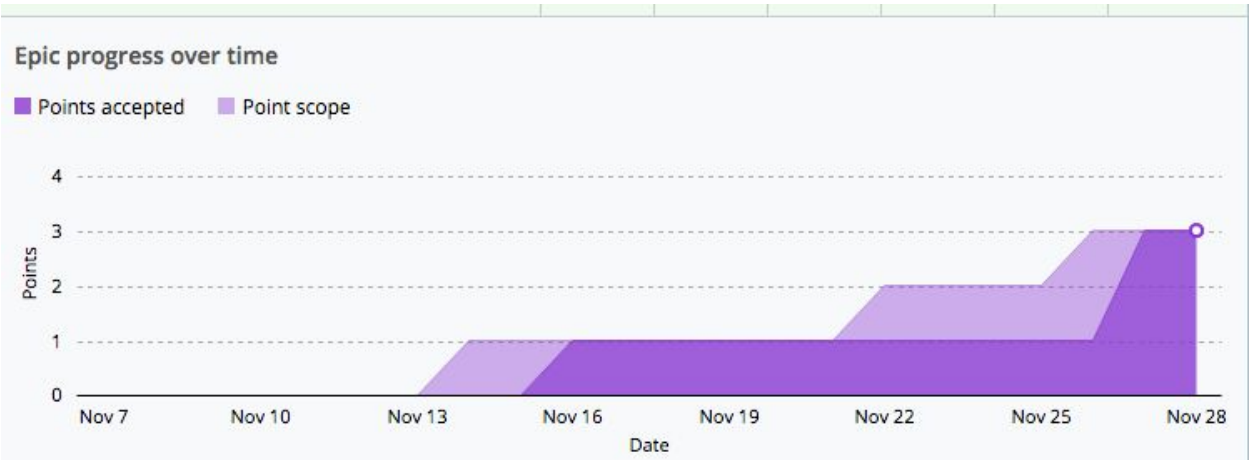
Sprint 4

The main focus of this sprint was finalizing rules and lake/road completing as well as implementing server communication.

SPRINT 4 BURNDOWN							
	Day						
Item(weight)	Monday 11/21	Tuesday 11/22	Wednesday 11/23	Thursday 11/24	Friday 11/25	Saturday 11/26	Sunday 11/27
Implement Tiger Placement	40	36	33	28	18	6	0
Implement Lake/Road Completion	13	13	12	10	10	5	0
Create Network Protocol	38	34	34	30	22	12	0
Finalize Road/Lake Completion	13	13	12	12	6	2	0
Finalize Scoring on Completion	15	15	13	13	10	5	0
Improve AI Efficiency	20	18	18	16	7	0	0

[Visualization of 4 Sprints]





6. Known Bugs/Issues

1. When implementing tile rotation and placement checking, there was a bug encountered where each tile of the same type had its edges shifted when rotated. For example, if a tile was rotated and placed, then later another tile of the same type was placed, this second tile would start with the shifted edges of the first tile. The source of this bug was due to how the tiles were created. Each tile is given a “sides” array and pushed to the stack. However, each tile of the same type has a pointer to the same sides array. Thus, when shifted, all the tiles of this type get shifted, which is unacceptable. This was resolved by cloning the array with a new memory allocation, so each tile has its array in a different space in memory.

2. Tiger Placements near lakes. Specifically tile in String format: “ JLJL- “ . There is no definite method to check whether or not the two lake sides are connected to other tiles; so the tiger placement made by the AI. To partially ameliorate the issue, if a til is contains a lake, make the assumption that the lake is connected to all other lake tiles around it--leaves a smaller room for error.

7. Resources

[1] Alistair Cockburn. (n.d.). Retrieved December 01, 2016, from http://alistair.cockburn.us/Hexagonal architecture

[2] Git Repo Link:

https://github.com/KenanY/tiger-zone

[3] Burndown charts:

https://drive.google.com/open?id=1K2yVaxa49pUtuKlKdLlXhDJRmtQuObB2QOvgnDTBXZo