UCS2504 – Foundations of Artificial Intelligence
Exercise 9: Case Study
Tsuro Game

Team Members:

Thirumurugan RA (3122235001149)
Vishal Muralidharan (3122235001162)

Table of Contents

1. Introduction

1.1. Overview

Tsuro, originally designed by Tom McMurchie, is a strategic tile-laying board game where players guide their markers along evolving paths. This software is a Python-based implementation of the game, developed with a focus on modularity, clarity, and adherence to object-oriented design principles.

The primary objective for a player is to be the last participant remaining on the game board. This is achieved by placing tiles to extend one's own path while strategically directing opponents' paths off the board.

1.2. Core Features

This digital adaptation includes the following key features:

- **Multi-Player Support**: Accommodates 2 to 8 players in a local, hot-seat turn structure.
- **Complete Tile Set**: Implements all 35 unique tiles from the original game, each with 4 rotational states, for 140 total configurations.
- **Game Board**: A 6×6 grid (36 cells) with 48 perimeter starting positions.
- **Automatic Path Following**: Markers automatically traverse paths created by tile connections.
- **Simultaneous Movement**: All affected markers move simultaneously after a tile is placed.
- **Elimination System**: Players are eliminated when their markers reach the board edge.
- **Dragon Tile Mechanic**: A system to manage tile distribution when the draw pile is depleted.
- **Placement Validation**: The system enforces game rules, such as requiring tile placement adjacent to the active player's marker.
- **Win/Tie Detection**: The game automatically identifies and announces a solo winner or a tie scenario.

2. Installation & Setup

2.1. Installation Steps (How to Run)

1. **Navigate to the project directory** using a terminal or command prompt.
   cd path/to/Tsuro-Final

2. **Create a virtual environment** (Recommended).
   # Windows
   python -m venv myenv
   # Linux/macOS
   python3 -m venv myenv

3. **Activate the virtual environment**.
   # Windows (Command Prompt)
   myenv\Scripts\activate.bat
   # Windows (PowerShell)
   myenv\Scripts\Activate.ps1
   # Linux/macOS
   source myenv/bin/activate

4. **Install dependencies** from the requirements.txt file.
   pip install -r requirements.txt

5. **Launch the game**.
   python main.py

3. Technical Details

3.1. Tech Stack

This project is built primarily with Python and the Arcade library.

- **Programming Language**: Python 3.10+
- **Graphics Library**: Python Arcade 2.6.17
- **Core Concept**: A graphics API (provided in arcadegraphics.py) is abstracted on top of Arcade to simplify rendering and event handling.

3.2. Core Algorithms & AI Concepts

Path-Following Algorithm

This game does **not** use a traditional dynamic pathfinding algorithm like A* or Dijkstra. Instead, it employs a deterministic **path-following mechanism** built on a dictionary-based lookup system. This is a form of pre-computed graph traversal.

How it Works:

1. Connection Dictionaries
Two dictionaries, _loc_dict1 and _loc_dict2, store the connections for all ports on the board. Since each path on a tile connects two ports, a single port location can be an entry point from two different directions, which is why two dictionaries are needed.

```
self._loc_dict1 = {}
self._loc_dict2 = {}
```

2. Tile Placement Updates Connections
When a tile is placed, the update_loc_dict method is called. It calculates the absolute coordinates for the tile's 8 ports (based on its center and rotation) and stores the path connections in the dictionaries.

```
def update_loc_dict(self, coord, center):
    """ Use coordinate from the tile, update location dictionary. """

    logic_list = [(-17, -50), (16, -50), (-50, -17), (50, -17)] +\
        [(-50, 16), (50, 16), (-17, 50), (16, 50)]
    for pairs in coord:
```

```
        location_a = tuple(map(sum, zip(center, logic_list[pairs[0] -
1])))
        location_b = tuple(map(sum, zip(center, logic_list[pairs[1] -
1])))

        # 2 dictionaries - every spot can lead to 2 other spots
        if self._loc_dict1.get(location_a, "no_value") != "no_value":
            self._loc_dict2[location_a] = location_b
        else:
            self._loc_dict1[location_a] = location_b
```

3. Marker Movement Algorithm

The move_markers method iterates through all active players. It checks if a player's current location is a key in either dictionary. If it is, the player is moved to the corresponding value (the connected port) and their location is updated. This repeats until the player's new location has no further connections.

```
def move_markers(self):
    for player in self._active_players:
        # Keep moving while there's a valid next position
        while self.moveable1(player) or self.moveable2(player):
            if self.moveable1(player) and not self.moveable2(player):
                new_loc = self._loc_dict1[player.return_current_loc()]
                player.move_player(new_loc)
                player.update_location(new_loc)

            if self.moveable2(player) and not self.moveable1(player):
                new_loc = self._loc_dict2[player.return_current_loc()]
                player.move_player(new_loc)
                player.update_location(new_loc)
```

4. Movement Rules & Summary

- The moveable1 and moveable2 methods check if the player's current location exists as a key in the respective dictionary.
- They also verify the destination has not been visited in the current move sequence (stored in the player's history) to prevent infinite loops.
- The marker follows this path greedily and deterministically.

This dictionary-based lookup is extremely efficient for Tsuro because all paths are fixed once a tile is placed, and players have no choice in which path to follow.

3.3. Architecture Highlights

- **Event-Driven Mouse Handling**: The system dispatches mouse events to the topmost clickable shape based on its "depth" (z-order), preventing clicks from "bleeding through" UI elements.
- **Dual-Dictionary Pathfinding**: (See section 3.2) Marker movement is handled by two dictionaries that map every connection point on the board, allowing for instant and deterministic path lookups.
- **Tile Rotation Logic**: Each tile's 4 rotational states are pre-defined in matches.txt. Right-clicking a tile in-game increments a rotation counter, and the game uses _rotations % 4 to select the correct connection pattern.

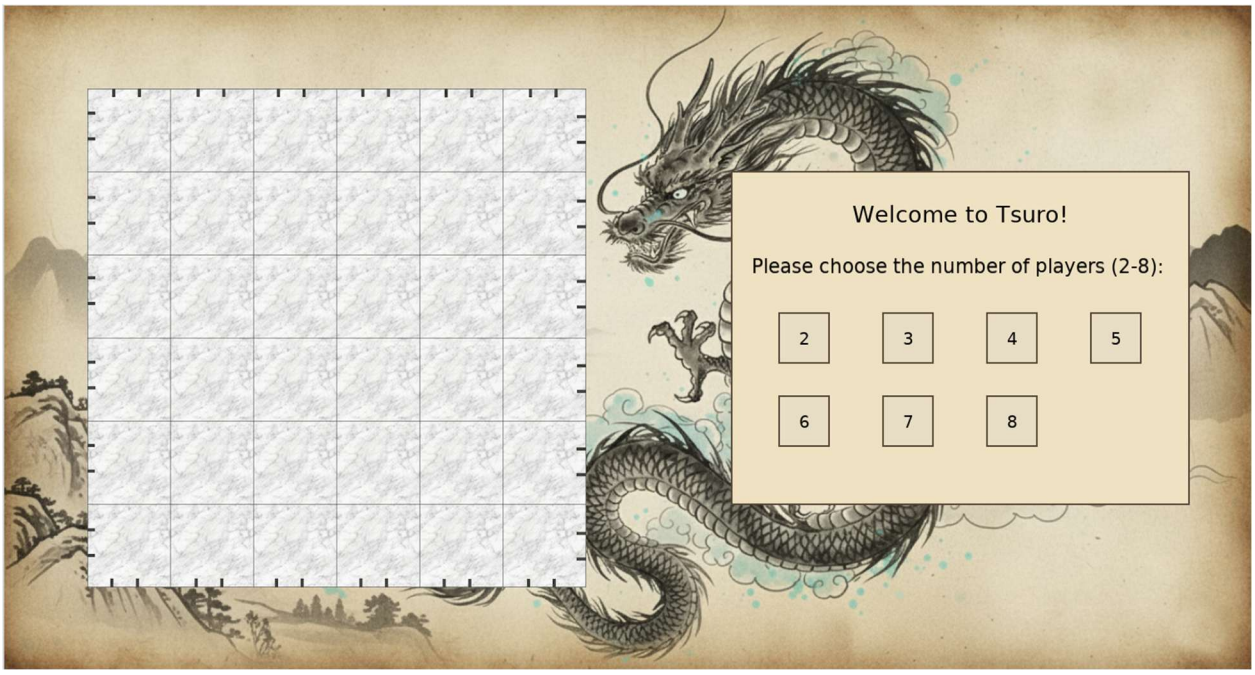
4. Game Mechanics & How to Play

4.1. The Goal

The objective of Tsuro is to be the **last player remaining on the board**. This is achieved by strategically placing tiles to extend one's own path while guiding opponents' paths off the board.

4.2. Setup Phase (Choosing Starting Positions)

1. **Select Player Count**: Upon launch, a popup prompts for the number of players (2-8).
2. **View Starting Hand**: Each player is dealt 3 random tiles, displayed on the right, to inform their starting decision.
3. **Choose Position**: Each player, in turn, interacts with a personalized prompt banner:
    - **Left click the prompt banner**: Cycles the player's marker clockwise to the next available starting position (48 total).
    - **Right click the prompt banner**: Confirms and locks in the selected starting position.
4. **Overlap Prevention**: The system prohibits starting at a position already occupied by another player.


Welcome to Tsuro!
Please choose the number of players (2-8):
2
3
4
5
6
7
8


2
1
Please view your starting tiles and choose a starting location!
Player 1, LEFT CLICK HERE to move your piece clockwise, RIGHT CLICK HERE to set your piece down.

4.3. Playing Your Turn

Each turn consists of three phases:

Phase A: Tile Selection

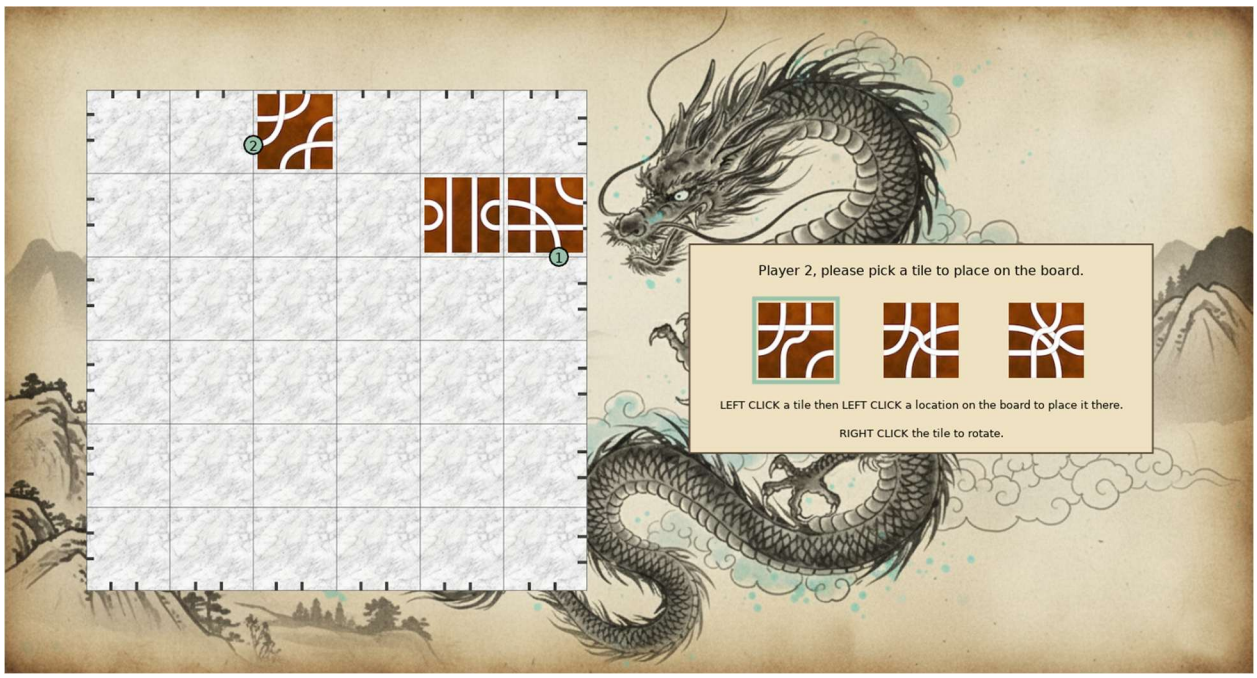
1. A prompt displays: *"Player X, please pick a tile to place on the board."*
2. The player's hand (up to 3 tiles) is shown on the right.
3. **Left click a tile** to select it. The selected tile is highlighted with a mint-green border.

Phase B: Rotation (Optional)

- **Right-click the selected tile** to rotate it 90° clockwise.
- The tile can be rotated multiple times to achieve the desired orientation.

Phase C: Placement

- **Left click an empty cell** on the 6x6 grid to place the selected tile.
- **Placement Rule**: The tile *must* be placed on one of the 8 cells adjacent (including diagonals) to the player's marker. Placement on an occupied cell is not permitted.




1
2
Player 2, please pick a tile to place on the board.
LEFT CLICK a tile then LEFT CLICK a location on the board to place it there.
RIGHT CLICK the tile to rotate.
WARNING: You can only place a tile next to your piece!


1
Please view your starting tiles and choose a starting location!
WARNING: You cannot place your piece here, this spot is taken
Player 2, LEFT CLICK HERE to move your piece clockwise, RIGHT CLICK HERE to set your piece down.

4.4. Automatic Movement

Immediately after a tile is placed, all player markers adjacent to the new tile automatically follow their connected paths. This movement continues until the path ends at an open port or the marker exits the board.

4.5. Tile Management & The Dragon Tile

- **Hand Replenishment**: At the beginning of a player's *next* turn, their hand is automatically refilled to 3 tiles from the draw pile, if available.
- **The Dragon Tile**: When the main draw pile is empty, the **Dragon Tile** is used.
  - The first player who needs to draw receives the Dragon Tile.
  - The Dragon Tile **cannot be placed**; it serves as a placeholder.
  - If another player is eliminated, their tiles are returned to the draw pile. The Dragon Tile holder will then automatically exchange it for a tile from the pile at the start of their next turn.

4.6. Elimination

A player is **eliminated** if their marker follows a path that reaches any of the 48 perimeter positions (i.e., it exits the board).

- **Exception**: A player's initial move from a starting position does not count as an elimination.
- When eliminated, the player's marker is removed, and any tiles in their hand are returned to the draw pile.

4.7. Victory Conditions

A winner is declared under the following conditions:

- **Win**: A single player is the **last player remaining** on the board. A victory message is displayed.
- **Tie**: If all remaining players (two or more) are eliminated on the *same turn*, the game ends in a tie.

5. Controls

This game is entirely mouse driven.

5.1. Mouse Controls

During Setup Phase

Action	Control	Description
Move Marker	**Left click** (on prompt banner)	Cycles marker clockwise around the perimeter.
Confirm Position	**Right click** (on prompt banner)	Locks in the starting position.

During Gameplay

Action	Control	Description
Select Tile	**Left click** (on a tile in hand)	Select the tile. Click again to deselect.
Rotate Tile	**Right click** (on any tile in hand)	Rotates the tile 90° clockwise.
Place Tile	**Left click** (on a board cell)	Place the currently selected tile onto the board.

6. Project Repository

View Project Repository on GitHub: https://github.com/thirumuruganra/Tsuro