Decentralized Tangram

A game of co-operative tangram on distributed systems

Dafang Cao (j2d0b)
Edward Zhou (u6x9a)
Max Wei (e3v8)
Michael Chen (e6y9a)
Stephanie Chan (g2u9a)

Table of Contents

Introduction

The *tangram* is a tiling puzzle game invented in China in the Song Dynasty (960-1279) and introduced to Europe in the early 19th century. It consists of seven flat shapes, called *tans*, and the objective of the puzzle is to put together the shapes without overlapping to form a new shape. In electronic form, it is a game where a player can select a shape, move it around and rotate it before placing it in its correct position.

Background

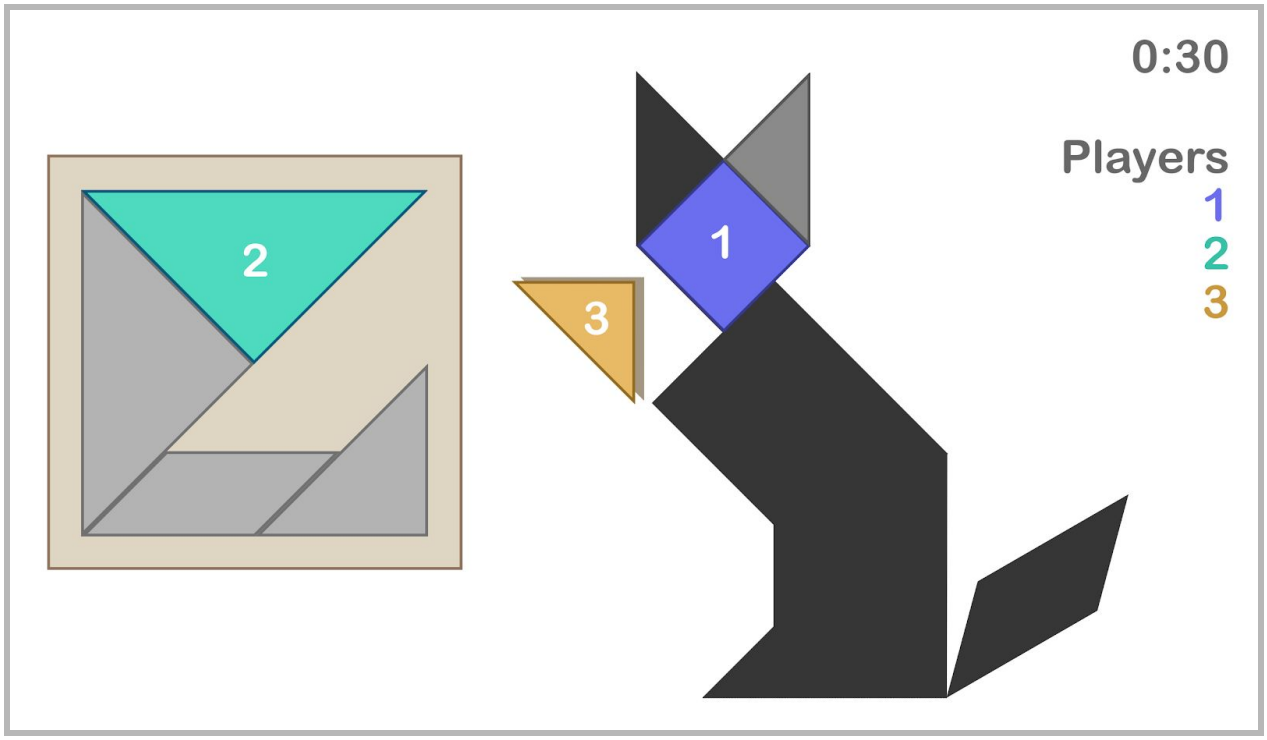
There are many implementations of tangram available online and they offer standard single player gameplay. Multiplayer co-op (where everyone is working together to solve the puzzle) is an aspect that we want to introduce into tangram because this will add complexity to the gameplay in terms of communication and coordination with other players. We have not found any multiplayer co-op versions of tangram, nor any implementation of the game in Golang, which indicates that a version of tangram with a multiplayer co-op feature has yet to be made.

In Decentralized Tangram, our multiplayer co-op tangram game, each player will have to start their own client to start or join a game. The game can have any number of players. Each player can control up to one tangram piece. If no pieces are available, a player must wait until another player releases control of a piece.

The game state is held in every single client and every time the client does something to change the game state, it has to communicate the changes with all of the other clients.

The game state consists of the following:

- The seven tangram pieces, ordered by last interaction, and their locations (x, y coordinate), rotation alignment (in increments of 5 degrees), and the player controlling the piece
- A timer
- A list of players
- The shape to make



(Figure 1) Mockup of Game with 3 Players.

System Topology

The system will be a fully connected graph, where each client is aware of every other client within the system. Clients, upon connection to any node, will receive information about the other participants of the game. Each collection of nodes represents one tangram game.

Clients may, for optimization reasons, opt to begin a game with a host. The host is dynamically selected based on merit; in this case, average latency between all other clients. The topology of this system will be a star. All clients will still be aware of other clients within the game, however they will not communicate except for in the case that they need to select a new host due to host failures.

Trust

The clients will be assumed to be trusted. That is, as the gameplay genre is co-operative puzzle and it is between individuals who have shared their IP and port to one another, the incentive for cheating is minimized.

Problems and Approaches

Clock Synchronization

Clocks across different clients need to be softly synchronized because of the timer and order of actions. We will use NTP for clock synchronization to determine the order of concurrent actions made by different players.

Solution Checking

A real tangram puzzle may have multiple solutions. For example, a square block can be replaced by two triangles. To simplify implementation, we assume that solutions are unique save for identical shapes. Therefore a puzzle is considered solved when all pieces fall into their designated place.

Client Behavior

- The player may specify, on client launch, the address of another player to join their game. If the game already has the maximum number of players connected, the client will default to a new game. Omitting the address on client launch will launch a new game.
- Clients maintain heartbeats to every client in the game.
- On every game tick, each player sends out the state for the the piece it is manipulating to all other players, and updates its UI with any updates from other players.

Concurrency Issues

- If 2 players attempt to manipulate the same piece at the exact same time, the game should resolve the winner in a reasonable amount of time with minimal visual anomaly. Our proposal is to send a timestamp alongside each event. If identical timestamps are found, both clients will send a random number, and the greater between the two wrests control of the piece.

Testing

We will manually test various functionalities of our game. Examples of behaviour to test include:

- Single player tangram
    - General behaviour
        - Movement

        - Selecting and moving pieces
    - Clients are able to connect to this tangram client via websocket protocol
- Multiplayer tangram
    - Actions from other clients should register for every client
    - Timers are synchronized with a small margin of error
    - Client disconnections during a tangram game should not stall the game
    - Client reconnections during a tangram game should be possible

SWOT Analysis

	Helpful	**Harmful**
Internal	Strengths 1. Max in our group has experience in making games 2. Edward has experience in websockets 3. Dafang has an awareness of potential problems with distributed multiplayer games	Weaknesses 1. Steph is new to the group so there is some uncertainty in group cohesiveness 2. All group members only started learning go at the beginning of this course 3. Nobody has worked with vector clocks before.
External	Opportunities 1. Currently there is no existing multiplayer co-op game for Tangram in Golang 2. Game systems are hardly decentralized. This is an opportunity to explore the topic of decentralized games.	Threats 1. Checking shape intersection and validation will be difficult unless we can find a specific library for it - Max will be looking at this as he worked on blockartlib intersections 2. We do not know how latency will affect the gameplay

Timeline

March 9th:

- Final Proposal due

March 13th:

- (Stephanie) Start creating data structures for game
- (Dafang) Implement function to nominate host (extra)

March 15th:

- Data structures due

- (Edward) Set up websocket connection between clients
- (Luxian) Flooding protocols for clients who are trying to update state
- (Max) Implement UI for the game and shape validation
- (Dafang) Time Sync Issues

March 21th:

- (Luxian) Send email to TAs to schedule a meeting on our progress
- Complete networking

March 26th:

- Time sync due
- Do testing

March 28th:

- UI due
- Start squashing all bugs

April 1st:

- All bugs should be fixed by today
- Start writing final report

April 6th:

- Final report and project code due

API

Client

- createGame(port int, initialState GameState) (err error)
    - Create a server which allows connections
- connect(address string) (client Client, state GameState, err error)
    - Connects to a server. Upon connection, allow others to connect to you to join the network
- update(state GameState) (newState GameState, err error)
    - Updates the game state given higher level actions
    - These may include:
        - Pick up tangram
        - Drop tangram
        - Move tangram
- heartbeat() (err error)
    - Checks to see if the client is alive
- getState() (state GameState, err error)
    - Allows UI to receive updates about the state
- disconnect() (err error)
    - Leave the game lobby, closing all websocket connections

References

https://en.wikipedia.org/wiki/Tangram
https://www.mathsisfun.com/games/tangrams.html