# Boop Game - Client-Server Architecture

This is a client-server implementation of the Boop board game where two players connect to a server to play against each other.

## Architecture Overview

- **Server** (`server.py`): Manages game state, validates moves, displays the game GUI, and coordinates between two clients
- **Clients**: Three different client implementations
  - `client_human.py`: Human player with GUI interface
  - `client_random.py`: AI that makes random valid moves
  - `client_ai.py`: AI using minimax algorithm with alpha-beta pruning

## Setup and Running

### 1. Start the Server

First, start the server which will display the game board and wait for two clients to connect:

```bash
python server.py
```

The server will:

- Start listening on `localhost:5555`
- Display a game window showing the board
- Wait for exactly 2 players to connect
- Begin the game once both players are connected

### 2. Connect Clients

You need to start exactly **2 clients** to play. You can mix and match any combination of the three client types.

#### Human Client (GUI)

```bash
```

```
# Connect to localhost:5555 (default)
python client_human.py

# Connect to a specific host and port
python client_human.py 192.168.1.100 5555
```

- Displays a full game GUI
- Click on piece icons (cat/kitten) to select piece type
- Click on the board to place pieces
- Only allows moves when it's your turn

**Usage:** `python client_human.py [host] [port]`

### Random AI Client

```bash
# Connect to localhost:5555 (default)
python client_random.py

# Connect to a specific host and port
python client_random.py 192.168.1.100 5555
```

- Runs in terminal (no GUI)
- Automatically makes random valid moves
- Useful for testing

**Usage:** `python client_random.py [host] [port]`

### Minimax AI Client

```bash
# Connect to localhost:5555 with depth 2 (default)
python client_ai.py

# Connect to a specific host and port with default depth
python client_ai.py 192.168.1.100 5555

# Connect to localhost:5555 with custom depth
python client_ai.py localhost 5555 3
```

- Runs in terminal (no GUI)
- Uses minimax algorithm with alpha-beta pruning
- Higher depth = stronger but slower
- Recommended depths: 1-3 (depth 4+ can be very slow)

**Usage:** `python client_ai.py [host] [port] [depth]`

## Example Game Configurations

### Human vs Human

```bash
# Terminal 1
python server.py

# Terminal 2
python client_human.py

# Terminal 3
python client_human.py
```

### Human vs AI

```bash
# Terminal 1
python server.py

# Terminal 2
python client_human.py

# Terminal 3
python client_ai.py 2
```

### AI vs AI

```bash
```

```
# Terminal 1
python server.py

# Terminal 2
python client_ai.py localhost 5555 2

# Terminal 3
python client_ai.py localhost 5555 3
```

### Random vs AI (for testing)

```bash
# Terminal 1
python server.py

# Terminal 2
python client_random.py

# Terminal 3
python client_ai.py localhost 5555 2
```

### Playing Over Network

```bash
# On server machine (e.g., 192.168.1.100)
python server.py

# On client machine 1
python client_human.py 192.168.1.100 5555

# On client machine 2
python client_ai.py 192.168.1.100 5555 2
```

## Game Rules

Players take turns placing kittens or cats on a 6x6 board:

1. **Placement**: Click/select a piece type, then place it on an empty cell
2. **Booping**: Placing a piece pushes adjacent opponent pieces (and same-color pieces)
    - Kittens can boop other kittens

- Cats can boop both kittens and cats

3. **Three in a Row**: Get 3 pieces in a row (with at least one kitten) → those pieces become cats
4. **Winning**:
   - Get 8 cats on the board, OR
   - Get 3 cats in a row (no kittens)

## Network Protocol

The server and clients communicate via JSON messages over TCP sockets:

### Server → Client Messages

- `assignment`: Assigns player number (0=Orange, 1=Black)
- `game_state`: Full game state update after each move
- `error`: Error message for invalid moves

### Client → Server Messages

- `move`: Request to place a piece at (x, y)

## Features

- **Move Validation**: Server validates all moves before applying them
- **Real-time Updates**: All clients receive game state updates immediately
- **Reconnection Handling**: Clients detect disconnections
- **Visual Feedback**: Server GUI shows last placed piece and current turn

## Customization

### Change Server Port

Edit `server.py`:

```python
port = 5555  # Change to desired port
```

Or when connecting clients, specify the new port:

```bash
```

```
python client_human.py localhost 6000
```

### Change AI Difficulty

For `client_ai.py`, specify the depth parameter:

```bash
python client_ai.py localhost 5555 4  # Harder but slower
```

### Connect to Remote Server

Clients can connect to a server on another machine:

```bash
# If server is running on 192.168.1.100
python client_human.py 192.168.1.100 5555
python client_ai.py 192.168.1.100 5555 3
```

## Troubleshooting

**Problem**: "Address already in use" error
**Solution**: Wait a few seconds for the port to be released, or change the port number

**Problem**: Client can't connect
**Solution**: Make sure the server is running first and the port matches

**Problem**: AI is too slow
**Solution**: Reduce the AI depth (use 1 or 2 instead of 3+)

**Problem**: Game freezes
**Solution**: Restart the server and reconnect clients

## File Structure

```
server.py          - Game server with GUI
client_human.py    - Human player with GUI
client_random.py   - Random move AI client
client_ai.py       - Minimax AI client
game.py            - Core game logic
ai.py              - AI implementation
gui.py             - Original GUI (used by human client)
```

| | |
|---|---|
| pieces.py | - Game piece definitions |
| constants.py | - Game constants |