CSCI201 Final Project: Detailed Design Document
Project Number 7: Game

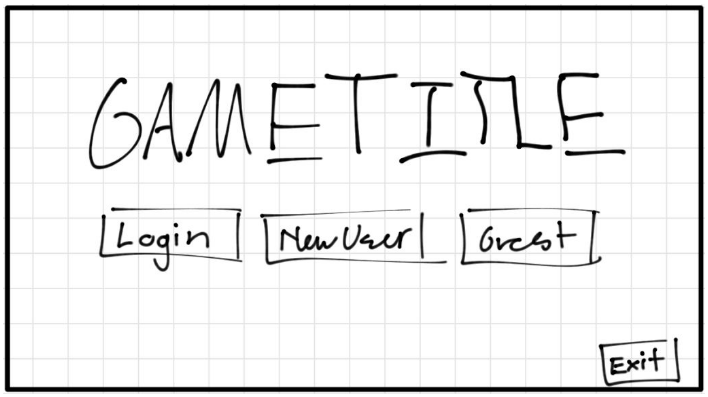
Team Members
Hayley Pike: hpike@usc.edu
Daniel Santoyo: dsantoyo@usc.edu
Yi(Ian) Sui: ysui@usc.edu
Ekta Gogri: egogri@usc.edu

Hardware/software requirements

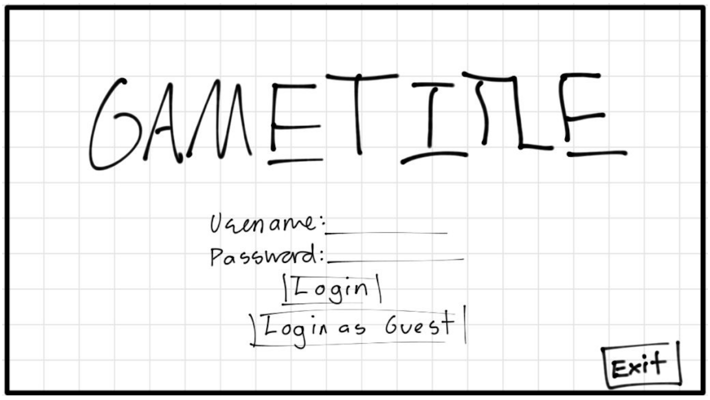
- User
    - Hardware:
        - RAM: 512 MB or more
        - Disk space: 512 MB or more
        - Processor: Minimum Pentium 2 266 MHz processor
        - Keyboard
        - Monitor with minimum 1024 * 768 resolution
        - Internet connection
    - Software:
        - Operating System: 64-bit
            - Windows: Windows 7 or newer
            - MacOS: 10.9 Mavericks or newer
        - A Java 8 or newer JRE/JDK
- Programmer
    - Hardware:
        - RAM: 512 MB or more
        - Disk space: 512 MB or more
        - Processor: Minimum Pentium 2 266 MHz processor
        - Keyboard
        - Monitor with minimum 1024 * 768 resolution
        - Internet connection
    - Software
        - Operating System: 64-bit
            - Windows: Windows 7 or newer
            - MacOS: 10.9 Mavericks or newer
        - A Java 8 or newer JRE/JDK
        - Eclipse Photon (4.8) or newer

GUI Mockup

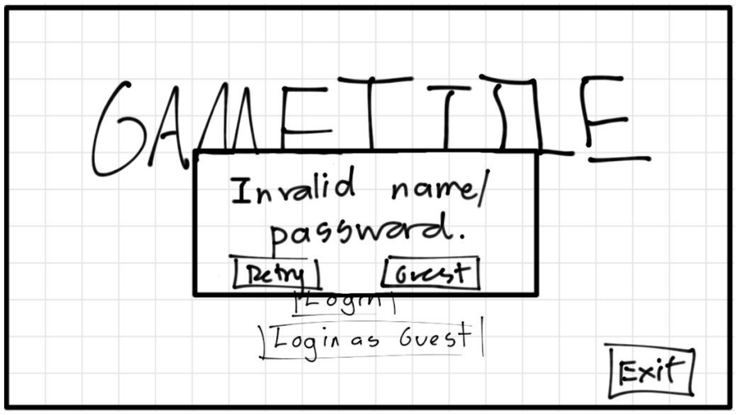
Starting Screen



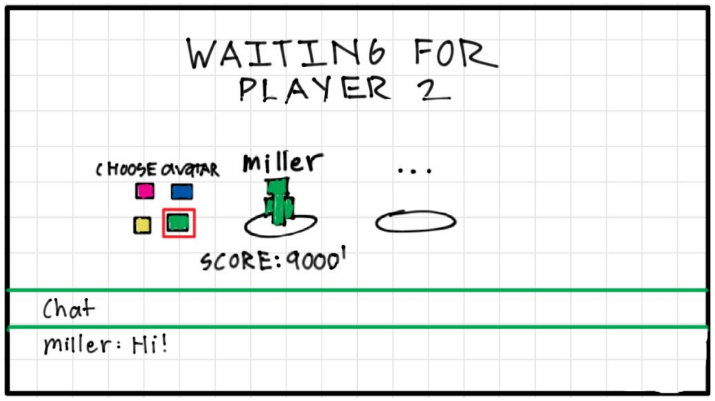
Login/Register Screen (pretty much the same thing)



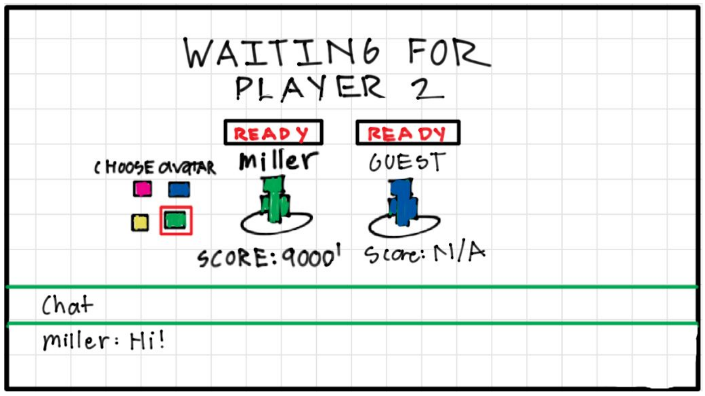
Invalid Login Screen



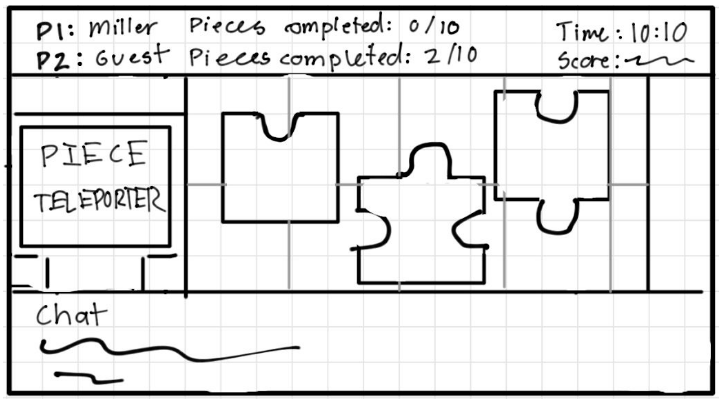
Waiting for Player Screen



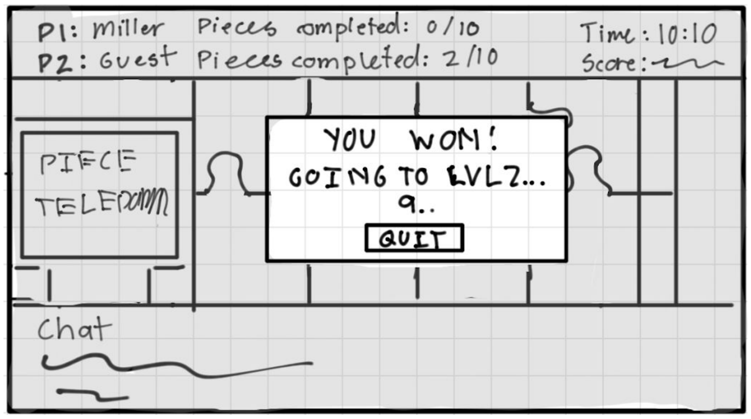
Ready to Start Screen



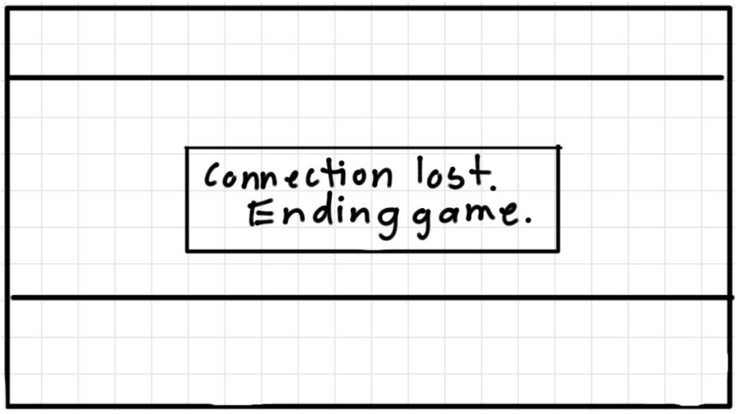
Main Game Screen (interact with mouse)


P1: miller Pieces completed: 0/10
P2: Guest Pieces completed: 2/10
Time: 10:10
Score:
PIECE TELEPORTER
Chat

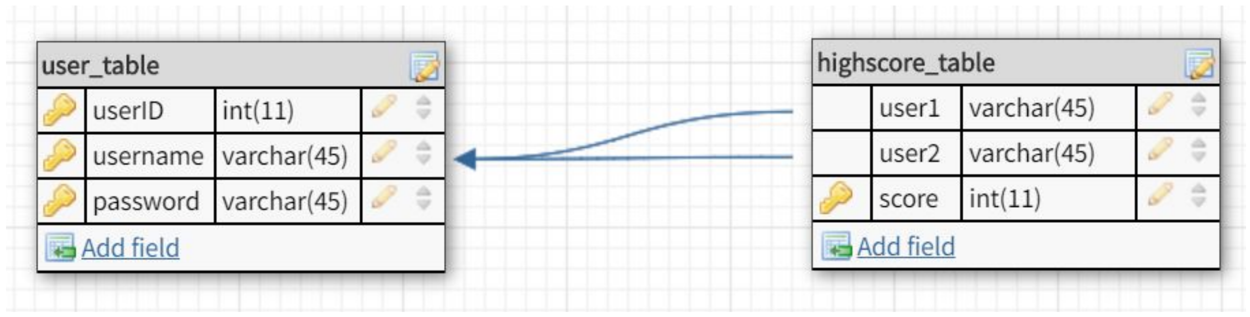
Level Transition Screen


P1: Miller Pieces completed: 0/10
P2: Guest Pieces completed: 2/10
Time: 10:10
Score:
PIECE
TELEROOM
YOU WON!
GOING TO LVL2...
9..
QUIT
Chat

Connection Lost Screen



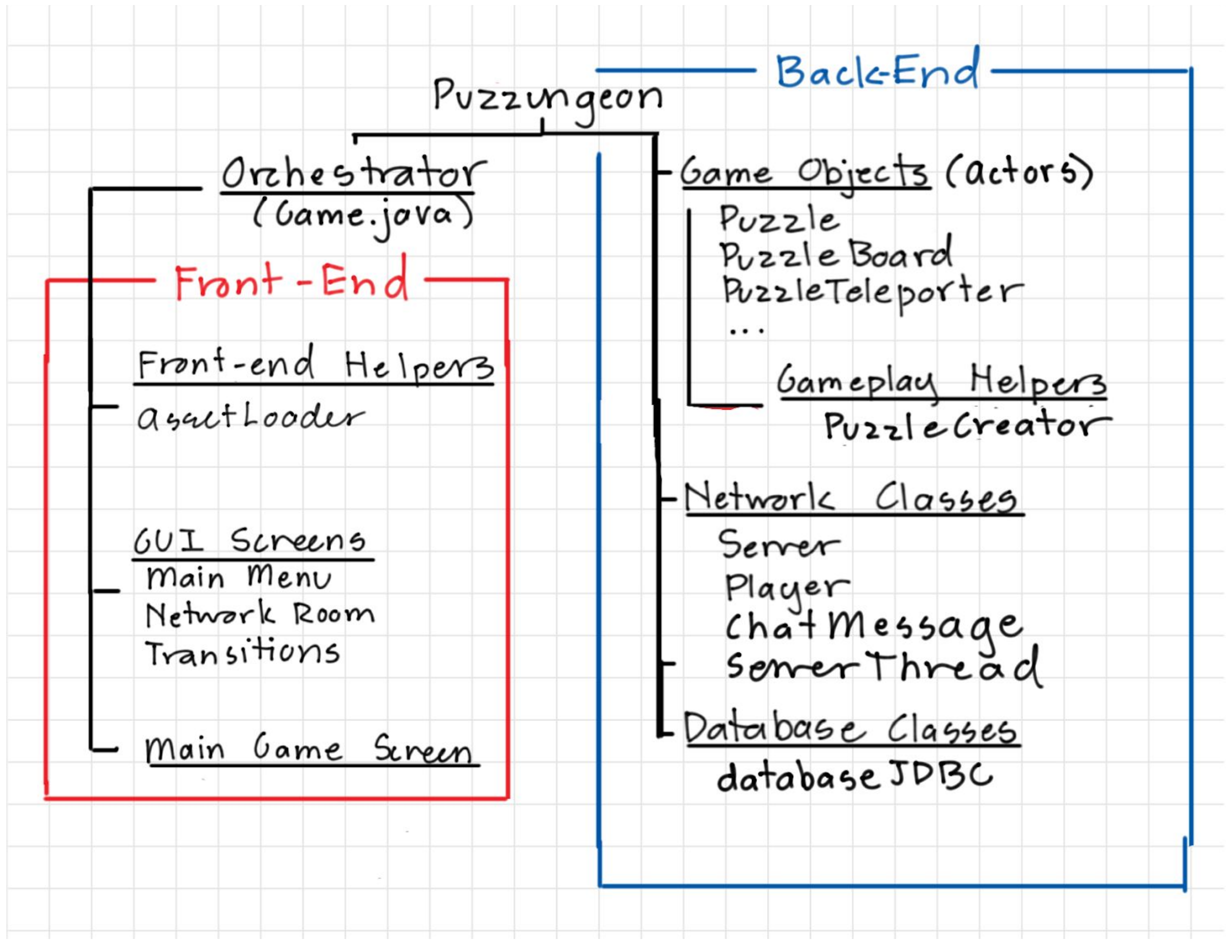
Database schema



user_table	
userID	int(11)
username	varchar(45)
password	varchar(45)

highscore_table	
user1	varchar(45)
user2	varchar(45)
score	int(11)

Class Diagram and Inheritance Hierarchy



Classes

Orchestrator

Class: Puzzungeon.java extends Game
Extends ApplicationListener (libGDX class), the core game class. Creates game and objects, runs render function, and disposes of assets upon program termination.

private Client client // connects to the server

//First screen. When new screens are needed they will be called inside the current Stage's method.
Public Stage mainMenu

Public void create() //loads assets and calls first screen

Public void dispose() //disposes visual assets in order to free up memory (called when switching screens and at end of program). Just a wrapper for AssetLoader's dispose().

Important method: setScreen(Stage stage)
//switches active Stage
//Each Stage object(com.badlogic.gdx.scenes.scene2d) handles part of the display of the
//game(input/image)

Asset Loader

Class: AssetLoader
Static class; loads text, image, and sound assets for all game objects.

Public static Skin uiSkin;
Public static TextureRegion puzzle1, puzzle2;
Public static BitmapFont titlefont, bodyfont, scorefont;
Public Sound music, click, pieceSuccess;
Public Texture background, puzzleTeleporter;

Public static void load() //loads all the assets

Public static void dispose() //clears all assets from memory. Called by Puzzungeon.dispose().

GUI Screens

Class: MainGameScreen implements Screen
Each Stage object(com.badlogic.gdx.scenes.scene2d) handles part of the display of the game(input/image)

Main Gameplay screen

Puzzungeon game; //reference to the game

Public MainGameScreen(puzzungeon game)
//sets game screen resolution, adds stage and its actors (game objects and UI), adds input processors, adds events listeners for UI

Public void act() //updates actors

Public void draw() //draws actors

Class: MainMenuScreen implements Screen
Each Stage object(com.badlogic.gdx.scenes.scene2d) handles part of the display of the game(input/image)

First screen; login/validation happens on this screen.

Puzzungeon game; //reference

Public MainMenuScreen() //sets game screen resolution, adds stage and its actors (game objects and UI), adds input processors, adds events listeners for UI

Public act() //updates actors

Public draw() //draws actors

Class: waitingRoom extends Screen
Each Stage object(com.badlogic.gdx.scenes.scene2d) handles part of the display of the game(input/image)

Network screen; network connection happens on this screen; server should let each client know if any other client is also connected

Puzzungeon game; //reference

Public waitingRoom() //sets game screen resolution, adds stage and its actors (game objects and UI), adds input processors, adds events listeners for UI

Public act() //updates actors

Public draw() //draws actors

Public void ifConnected()
//use a thread to continuously check if another client has connected to the server

Class: startGame extends Screen
Each Stage object(com.badlogic.gdx.scenes.scene2d) handles part of the display of the game(input/image)

Transitional screen; no input from user

Puzzungeon game; //reference

Public startGame() //sets game screen resolution, adds stage and its actors (game objects and UI), adds input processors, adds events listeners for UI

Public render() //updates and draws Actors in Stage

Class: levelTransition extends Screen
Each Stage object(com.badlogic.gdx.scenes.scene2d) handles part of the display of the game(input/image)

Transitional screen; user can choose new level or quit the game

Puzzungeon game; //reference

Public levelTransition() //sets game screen resolution, adds stage and its actors (game objects and UI), adds input processors, adds events listeners for UI

Public render() //updates and draws Actors in Stage

Class: endGame extends Screen

Each Stage object(com.badlogic.gdx.scenes.scene2d) handles part of the display of the game(input/image)

Transitional screen; appears when game is over, player can exit the game.

Puzzungeon game; //reference

Public endGame() //sets game screen resolution, adds stage and its actors (game objects and UI), adds input processors, adds events listeners for UI

Public render() //updates and draws Actors in Stage

Class: networkError extends Screen
Each Stage object(com.badlogic.gdx.scenes.scene2d) handles part of the display of the game(input/image)

Transitional screen; appears when network connection is lost. Player can only exit game.

Puzzungeon game; //reference

Public networkError() //sets game screen resolution, adds stage and its actors (game objects and UI), adds input processors, adds events listeners for UI

Public render() //updates and draws Actors in Stage

Public draw() //draws actors

Gameplay Objects

Class: Puzzle extends Image
Stores game object assets, input events

@Override private Vector2 position;

Private boolean inPosition;

Actors (Puzzle extends the libgdx Image class, which is an Actor that can be moved and placed in position)

public void displayPieces(): displays each scrambled piece as an individual actor

public void movePiece(): if not in correct position, move the pieces by implementing the “drag and drop” class
public bool checkPiece(): once an actor is moved, check if it is within bounds of its correct place on the puzzle board or the puzzle teleporter. If in correct place, lock in place. If within teleporter bounds, call sendPiece(). If neither, do nothing.
public boolean checkPuzzle(): once all pieces are in place, we wait for the other player or redirect to the game over screen
public void sendPiece(): if in bounds of the Teleporter, send piece(actor) to the stage of the other player

Class: PuzzleBoard extends Image
Stores game object assets, input events

@Override private Vector2 position;

Actors (PuzzleTeleporter extends the libgdx Image class, which is an Actor that can be moved and placed in position)

//(no direct input handling; handled in Puzzle class)

Class: PuzzleTeleporter extends Image
Stores game object assets, input events

@Override private Vector2 position;

Actors (PuzzleTeleporter extends the libgdx Image class, which is an Actor that can be moved and placed in position)

//(no direct input handling; handled in Puzzle class)

Server Classes

Class: chatMessage
each instance stores a message and the username of the sender

public static final long serialVersionUID
private String username
private String message

public ChatMessage(String username, String message) public String getUsername()

public String getMessage()

Class: ServerThread (extends Thread)
When a client object connects to the server, the server assigns it to a serverthread object
This object is used to handle data transfer between the server and the client

Networking Logic:
private ObjectInputStream ois
//catch input from the client class and send to the server

private ObjectOutputStream oos
//send data from the server to the client

private Server cs
// each ServerThread is linked with a server object

public ServerThread(Socket s, Server cs) // constructor
public void run()

(chatbox logic)
public void sendMessage(ChatMessage cm)
 //send chat message from the server to the client

Other game logic:

Puzzle puzzleOject
//Puzzle object used to call the sendPiece() method to send puzzle pieces between the stages of clients

Class: Server
back-end server. Should be combined with any other back-end logics

Network logic:
private Vector<ServerThread> serverThreads
// each ServerThread represents a client

public Server(int port)
// constructor. Initialize messageVec and serverThreads.
// assign new connection from a client object to a serverthread

(chatbox logic)
private Vector<ChatMessage> messageVec

// store chat message from every client

public void broadcast(ChatMessage cm)
// sending the newest message in server’s messageVec to every client

Game Objects:

Game Helpers:
createPieces(): splits the image into individual pieces
scramblePieces(): scrambles the pieces randomly and divides them between both users

public void clientsToServer()
//let each client know how many clients are connected to the server. For waiting room stage

Class: Client
store in-game data and connect to the server

private String username
private String password
private int userID
private int score

public Client(String hostname, int port)
//constructor. initialize messageVec. set up connection with the server. Read client information from the server

Network related:
private Socket s;
private ObjectInputStream ois;
private ObjectOutputStream oos;
private int port;
private String hostname;

chatbox logic
private Vector<ChatMessage> messageVec
//stores chat message sent from the server and is used to can display the messages

Database Classes

Class: DatabaseJDBC
Database logics: Create and store usernames/passwords

public bool loginValidation(String username, String password)
//Validates the user login by checking the matching of username and password

public int getHighScore()
//returns the high score of the client.

public bool ifGuestExists(String username)
//Checks if a second player is created as guest

public void addSecondPlayer(int limit)
//Creates player 2 via a similar method of creating player 1

Connection conn //Used to establish a connection
Statement st //Creates a statement and sends it to the database
ResultSet rs // Access certain keys from table(s) in the database
PreparedStatement ps //References a specific column of a table from the database
Scanner scan //Read Strings

//The ID from the player that gets stored/accessed via the ResultSet object in the database
Private int ID
//Similar to the above: Client score to be stored/accessed via the ResultSet object
Private int score
//Similar to the above: Client username to be stored/obtained via the ResultSet object
Private int limit
//Set limit to 2 so no more than two players can be created for a session.
Private String username
//Similar to the above: Password for the specific Client username stored/obtained via the ResultSet
Private String password