COP4331 Object Oriented Design and Programming

3 Dimensional Tic-Tac-Toe Game

Group Number: 23

Professor: Ionut Cardei

Team Members:
Bryan Barreto, Abir Faisal, Jamahl Farrington, Ruth Jimenez

Deliverable 3
Due: 12/05

Table of Contents

Application Requirements

@Authors Abir Fasial, Byran Barreto
Scenario: User Application, the software is used by end users as a desktop GUI application.

Use Case 1 – Application – Single Player

Description: Application is launched, and a user plays against a computer algorithm.

Actors: User
Preconditions:
   a. User is present
   b. Graphical User Interface (GUI) is available
   c. Application is running
Flow:
   1. Application presents options to the User
   2. User selects the ‘Single Player’ option
   3. Application assigns a SinglePlayerController Class to the GUI
   4. User plays against an algorithm implemented by the Class
   5. Application checks if there is a winner/tie after each move.
Terminations:
   a. User Wins
         1. Application informs User that it has won the game
         2. Application asks User if they would like to play again
   b. Computer Wins
         1. Application informs User that it has lost the game
         2. Application asks User if they would like to play again
   c. Game is tied
         1. Application informs User that the game is tied
         2. Application asks User if they would like to play again

@Authors Abir Fasial, Byran Barreto
Use Case 2 – Application – Single Client Host

Description: Application is launched and used as a host for a single client to join as player 2.

Actors:
   a. User1
   b. User2
Preconditions:
   a. User1 is present
   b. User2 is present
   c. User1 is connected to a network
   d. User2 is connected to a network
   e. Graphical User Interface (GUI) is available
   f. Application is running
Flow:
   1. Application presents options to the User1
   2. User1 selects the ‘Host Game’ option
   3. Application assigns a SingleHostController Class to the GUI
   4. Application displays connection information and waits for another user to join it
   5. User2's instance of Application presents options to the User2
   6. User2 selects the ‘Multiplayer’ option
   7. User2 enters connection information related to User1's instance.
   8. Application assigns a MultiPlayerClientController Class to the GUI using the connection information.
   9. User1 plays against User2 over a computer network
   10. Host Application (User1) checks if there is a winner/tie after each move.

Terminations:

d. User1 Wins
    1. Application informs User1 that it has won the game
    2. Application informs User2 that it has lost the game
    3. Application asks User1 and User2 if they would like to play again

e. User2 Wins
    1. Application informs User2 that it has won the game
    2. Application informs User1 that it has lost the game
    3. Application asks User1 and User2 if they would like to play again

f. Game is tied
    1. Application informs User1 and User2 that the game is tied
    2. Application asks User1 and User2 if they would like to play again

@Authors Abir Fasial, Byran Barreto
Use Case 3 – Application – Multiplayer Client

Description: Application is launched and used as a client and joins a host.

Actors: User, Server
Preconditions:
   a. User is present
   b. Server is present
   c. Graphical User Interface (GUI) is available
   d. Application is running
Flow:
   1. Application presents options to the User
   2. User selects the ‘Join Server’ option
   3. The application allows the user to connect to the default server or a server or host of their choice.
   4. Application connects to the server as a client.
   5. Server matches User with another client (User2).
      a. If there are no other players on the server, the client will play against an algorithm implemented by the server.
      b. If there is an odd number of of players, the client will play against an algorithm implemented by the server.
   6. Server tracks each set of players and the state of each game.
   7. Server ingests actions from each client.
   8. Server interprets the state of each game and informs clients of wins, losses, and ties.

Terminations:
   a. User disconnects
      i. The server will attempt to match the User with a different User.
   b. Server(host) disconnects
      i. The clients will attempt to recover the connection.
   c. Server runtime error
      i. The server will restart.

@Authors Abir Fasial, Byran Barreto
Scenario: Server. The software is used in a server environment to host players as clients.

Use Case 4 – Server – Multiplayer Host

Description: Program is run as a game server and used as a host.

Actors: Admin

Preconditions:

a. Command line is available

Flow:

1. Program is launched in server mode using the --server command line option.
2. Program presents itself as a server on the network
3. A number of players connect to the server
4. Server matches players
    a. If a player cannot be matched it will play against an algorithm implemented by the program.
5. Server tracks each set of players and the state of each game
6. Server ingests actions from each client and informs corresponding clients of said actions clients.
7. Server interprets the state of each game and informs clients of wins, losses, and ties.

CRC Cards

@Authors Abir Fasial

class, responsibilities, collaborators

Main	
load settings launch App or Server	App Server SettingsManager

package app/

App	
setup the main window initialize the MVC	Model View Controller

abstract class Model	
manage data for a view notify view of changes	View

abstract class View	
user input notify controller of input	Controller

abstract class Controller	
update the model	Model

package startscreen/

StartScreenModel extends Model	
define data keys store data values store server info	View

StartScreenView extends View	
show the start screen user selects game type user inputs server if needed	StartScreenController

StartScreenController extends Controller	
tells App what type of game to launch handle user interaction	App

package game/

GameModel	
define data keys store data values store gamestats	GameView

GameView	
define UI elements	GameControllers

enum GameType	
Provide strategy pattern Single,multi, or host game	App

SinglePlayerGameController	
run game internally handle game logic handle game input	GameView GameModel

MultiPlayerClientController	
handle game input send input to server receive from server	GameView GameModel

MultiPlayerHostController	
handle game input provide server for other player handle game logic	GameView GameModel

package util/

Solver	
provide a solver for the game	GameControllers Server

SettingsManager	
Load settings from file Save settings to file	Everything

package server/

Server	
provide a way for clients to connect and communicate	Clients

package chat/

ChatModel	
hold chat messages	

ChatView	
Display chat messages input new messages	

ChatController	
Handle the Chat window	ChatBotController ChatClientController

ChatBotController	
respond to chat messages in single player mode	

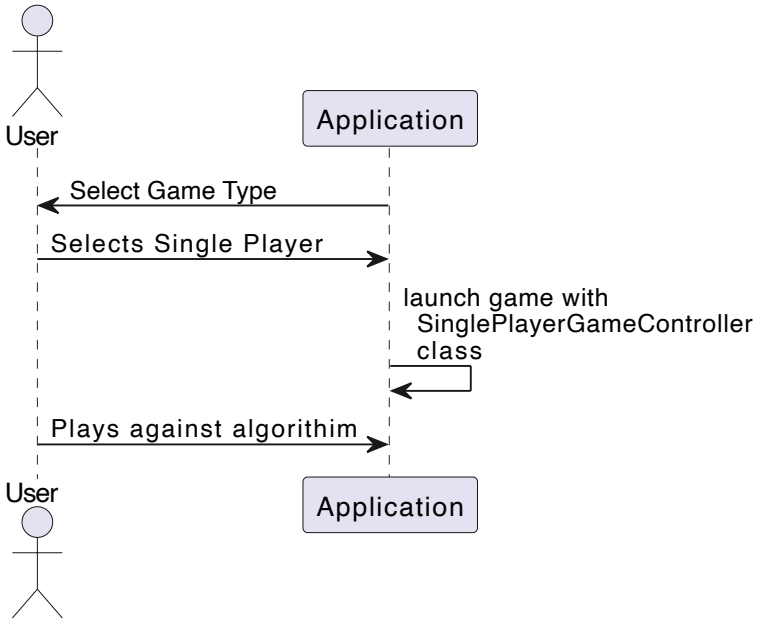
ChatClientController	
send receive messages from server	

Sequence Diagrams for scenarios

Description: Sequence diagrams for our use case scenarios

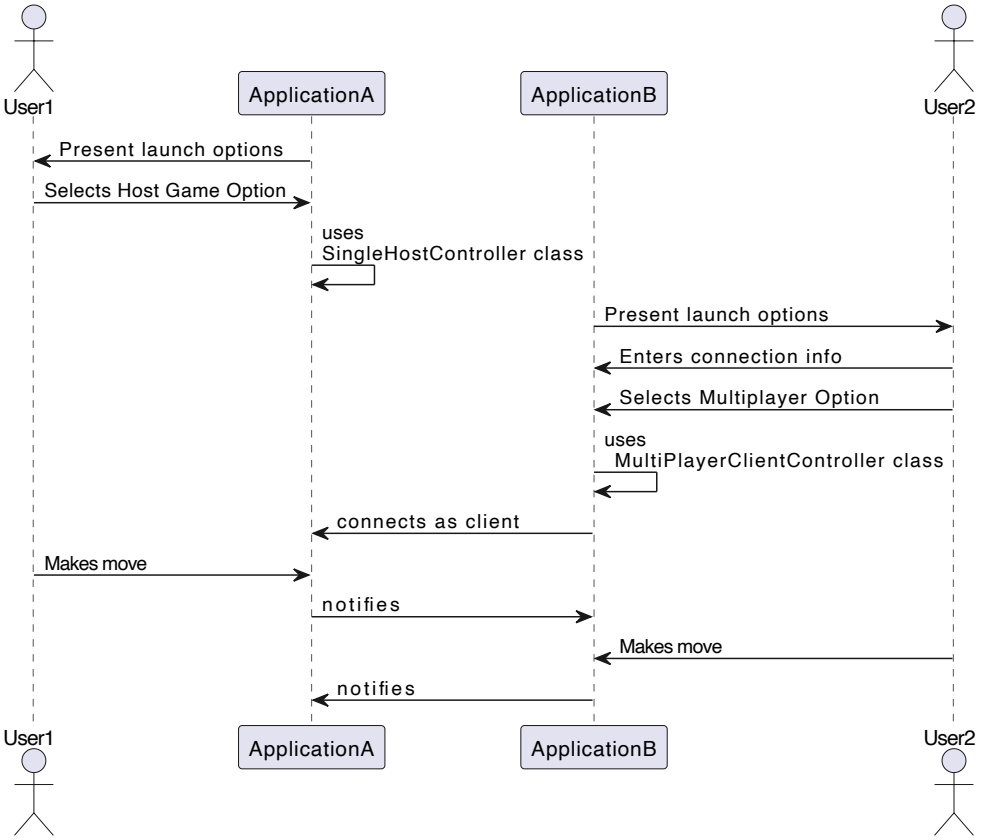
Author: Abir Faisal

Use Case 1 - Application - Single Player



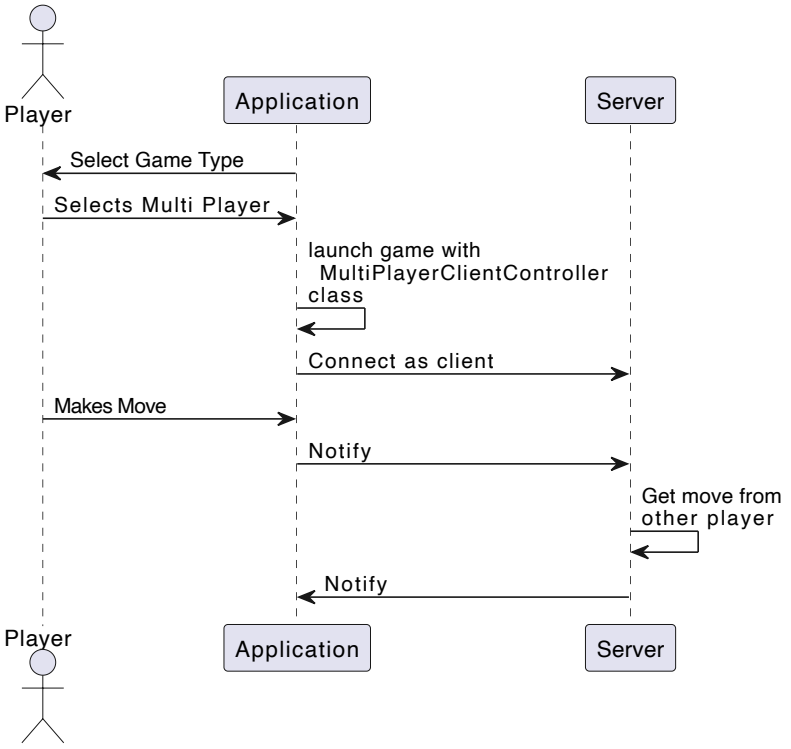
Author: Abir Faisal

Use Case 2 - Application - Single Client Host



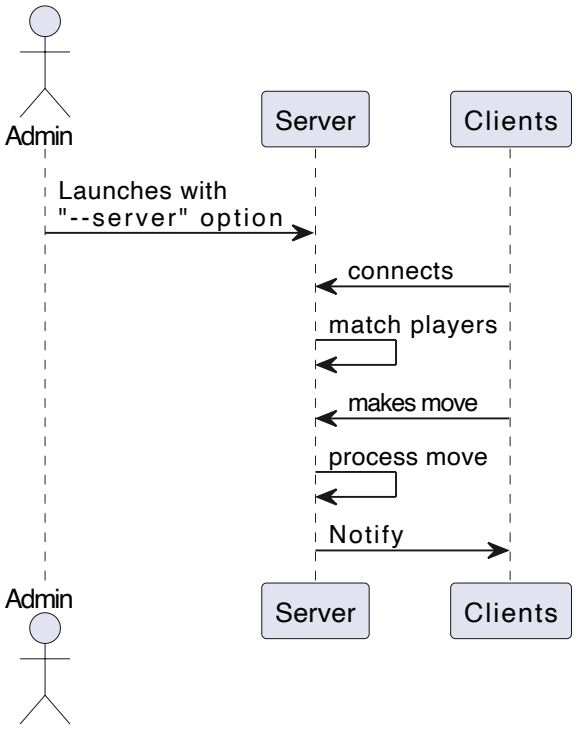
Author: Abir Faisal

Use Case 3 - Application - Multiplayer Client



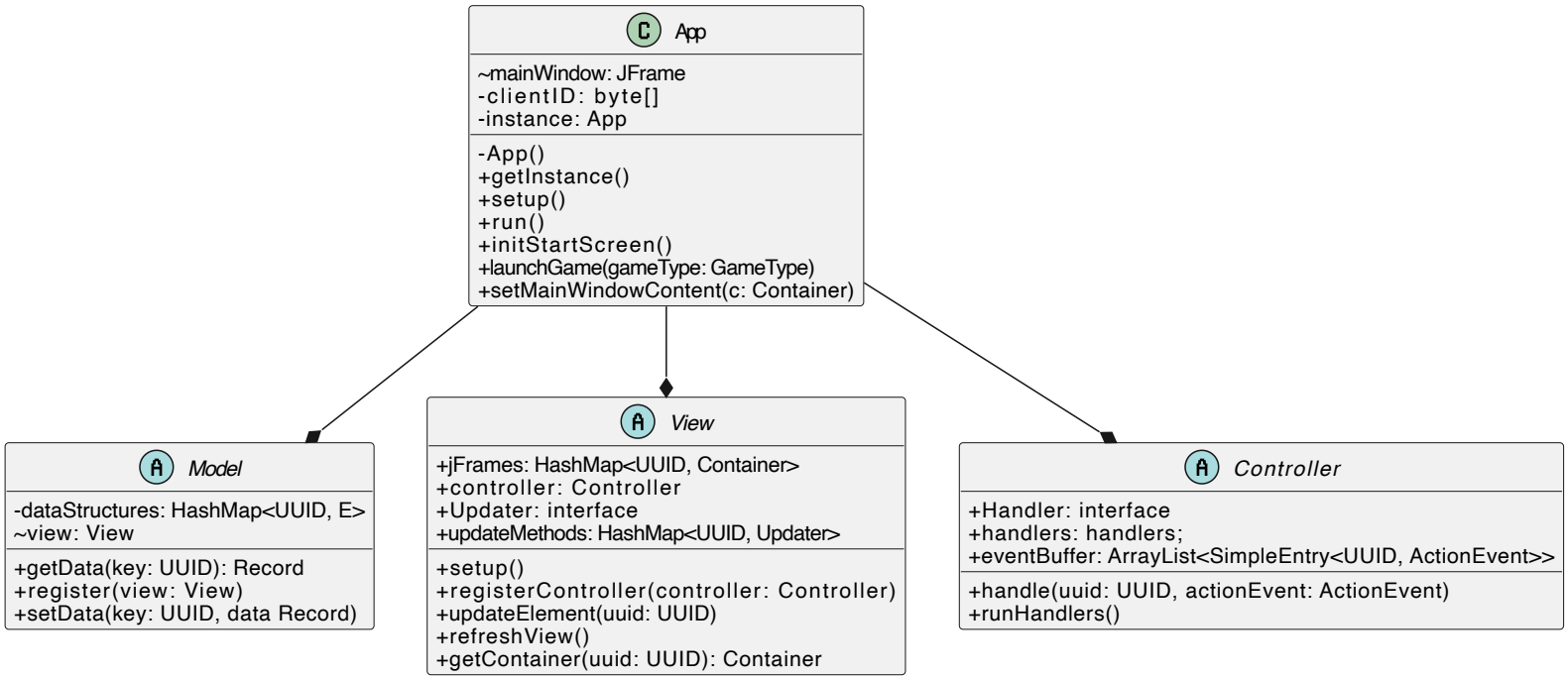
Author: Abir Faisal

Use Case 4 - Server - Multiplayer Host



UML Diagrams

Author: Abir Faisal

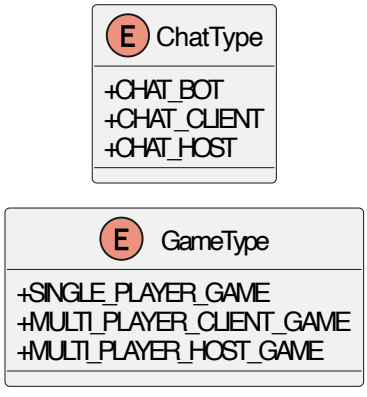


Description: The App class is implimented as a singleton pattern

It declares a Model, View, and Controller.

When the application is launched it initializes an empty window and puts whatever View type the programmer specifies into the mainWindow JFrame

More detail on next page

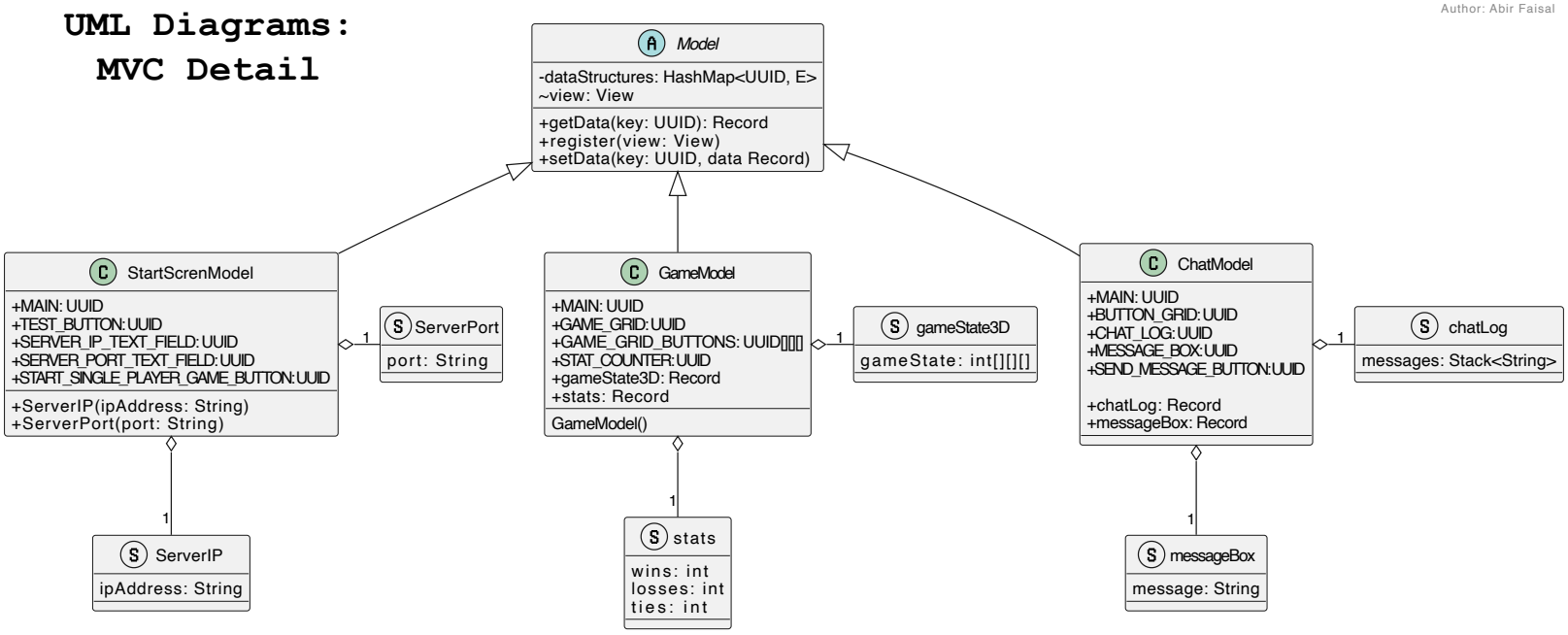


Description: These Enums are used by the launch controller to tell the App what type of Game and Chat to launch.

This is a strategy pattern.

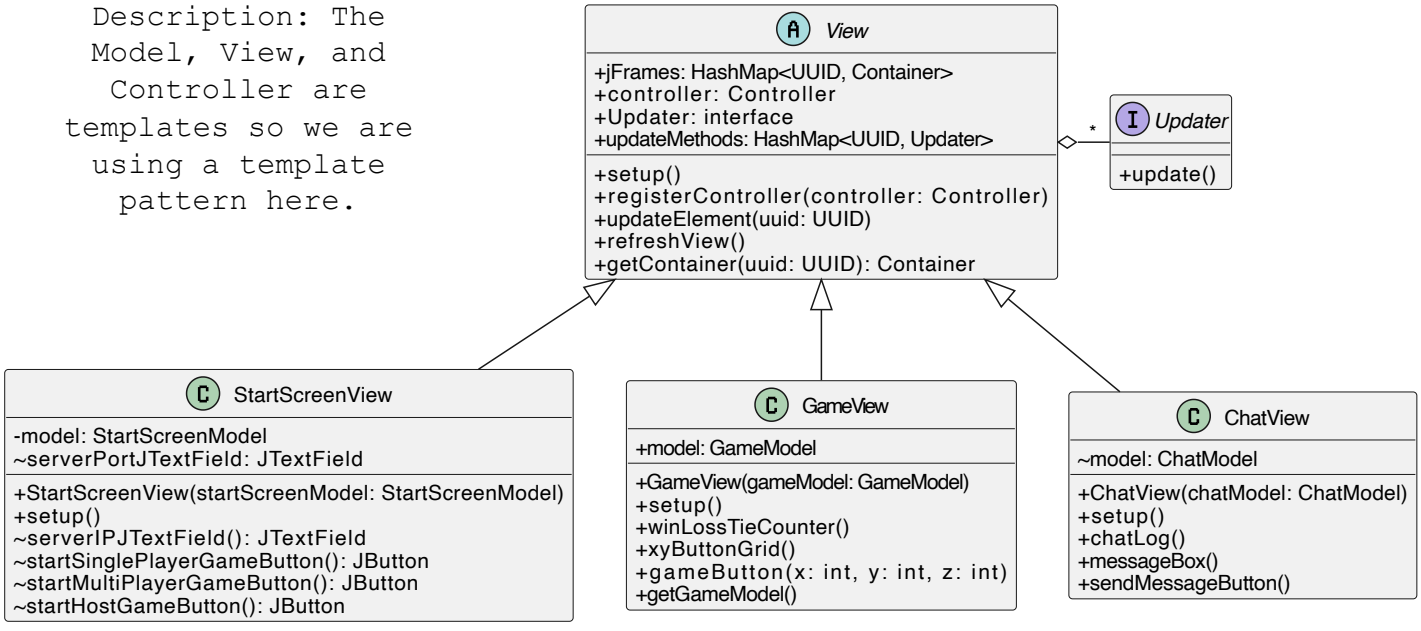
Author: Abir Faisal

UML Diagrams: MVC Detail



Author: Abir Faisal

Description: The Model, View, and Controller are templates so we are using a template pattern here.

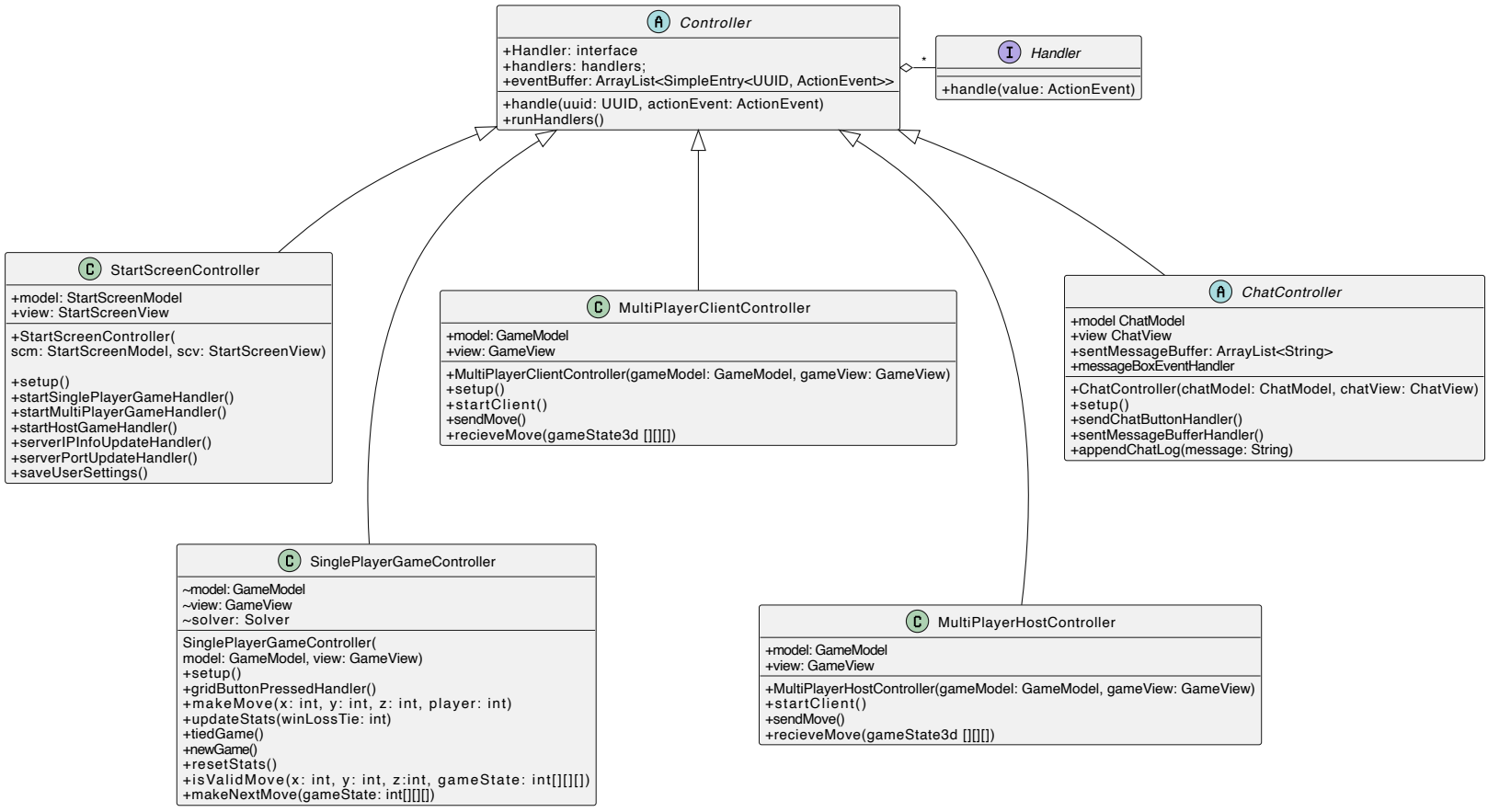


Controller Detail on next page

UML Diagrams:
MVC Detail

Description: The Model, View, and Controller details continued

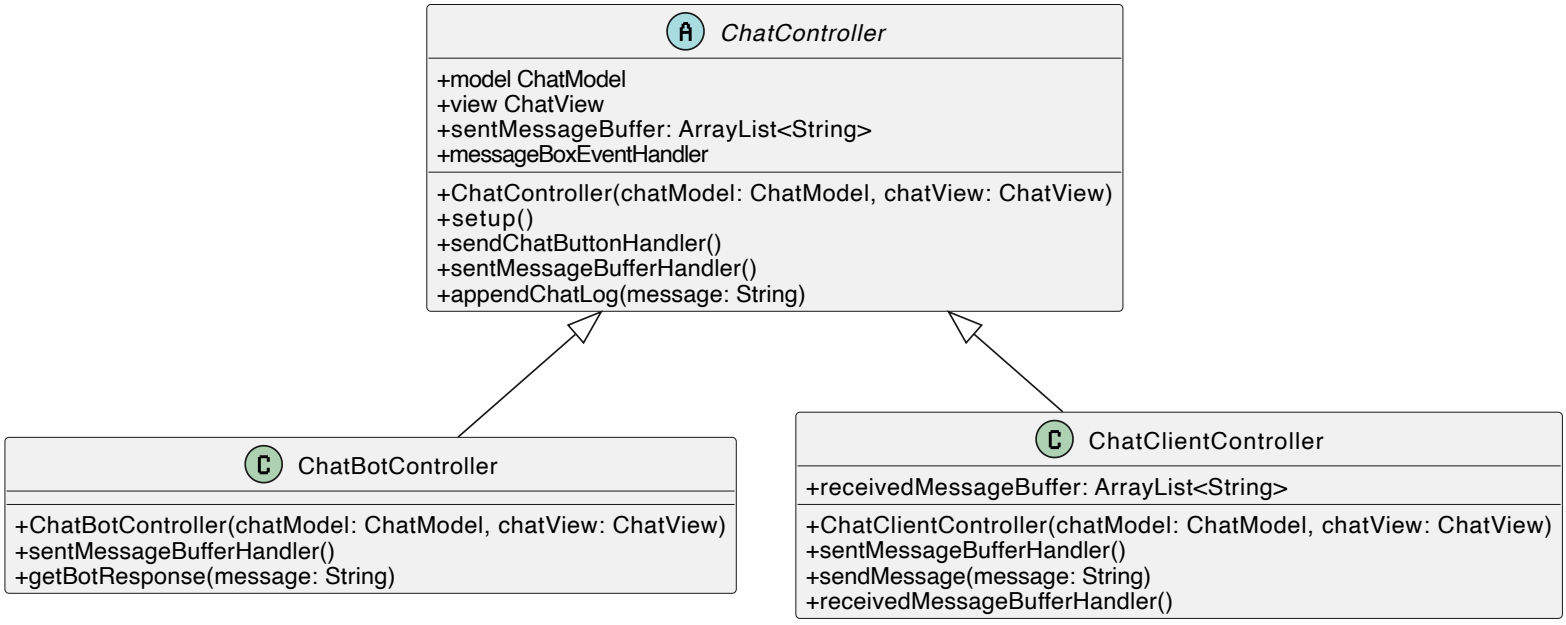
Author: Abir Faisal



UML Diagrams: MVC Detail
Chat Feature

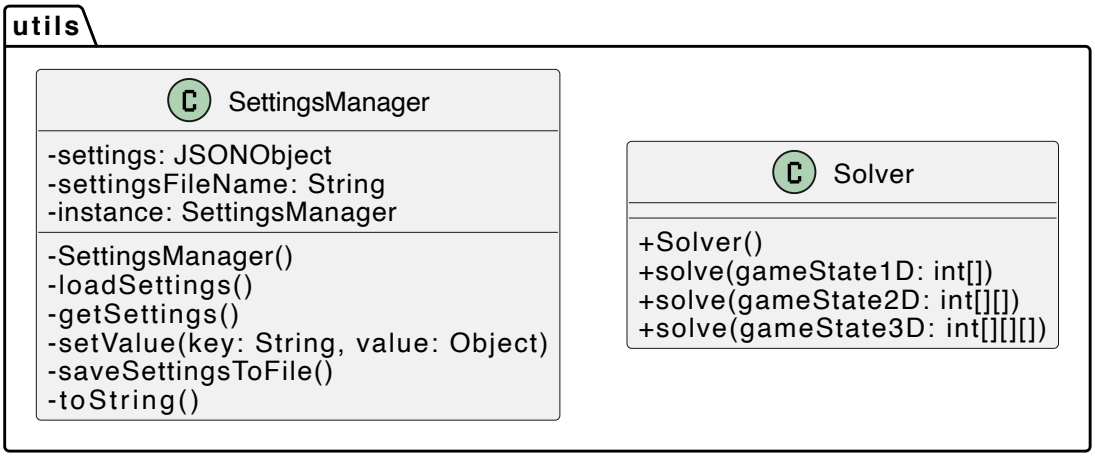
Description: The chat feature was not a part of our original design, but we thought it would show how our MVC architecture is extensible, you can just extend the MVC classes and make your own thing. It doesn't have to be a game it can be anything.

Author: Abir Faisal

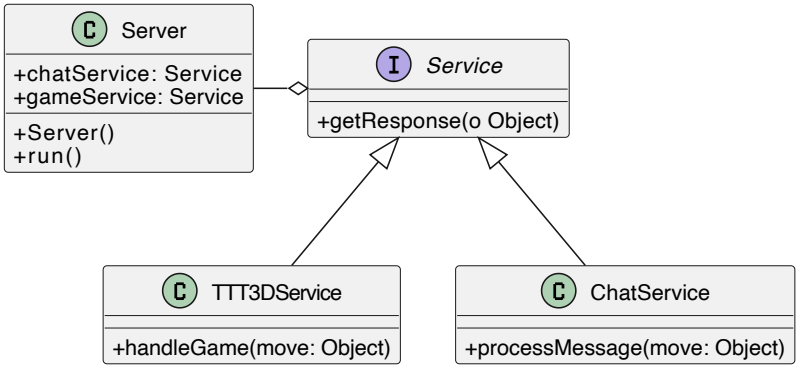


UML Diagrams: Other

Authors: Abir Faisal



Description: These are utilities that various parts of the program can use as needed.
The solver can check the game for a winner, and the SettingsManager loads and saves program settings from file.

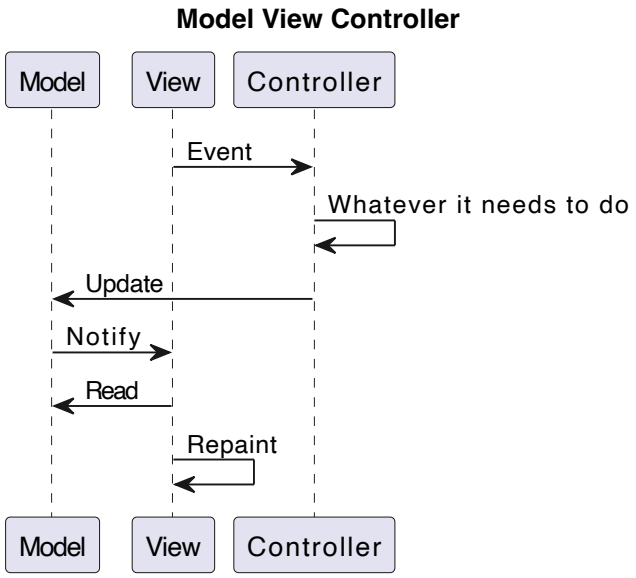


Description: The server is suppose to provide services for clients.

We didn't get to it but basically the idea was that the server could provide any service to any client as long as there exists a service handler.

Sequence Diagrams for program

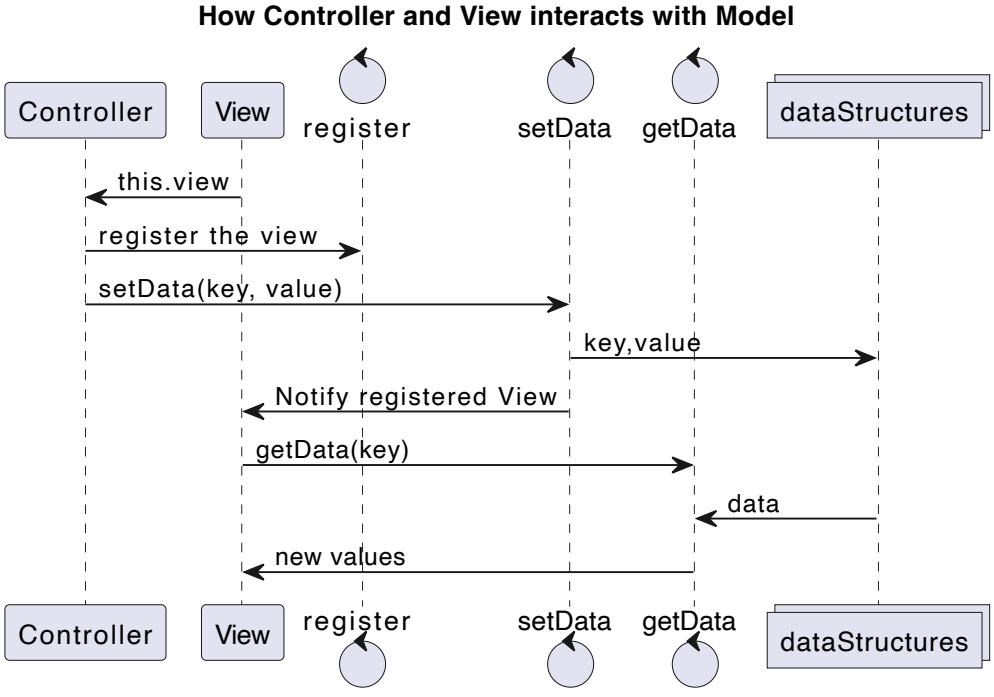
Author: Abir Faisal



Description: The MVC components interact with each other using UUIDs that are defined in models that extend the abstract Model. (Template pattern)

The templates contain everything needed forthe MVC to work. You just have to define the UI components in your View subclass, event handlers in your Controller subclass, and UUID constants in the Model as well as data strctures in the form of record classes.

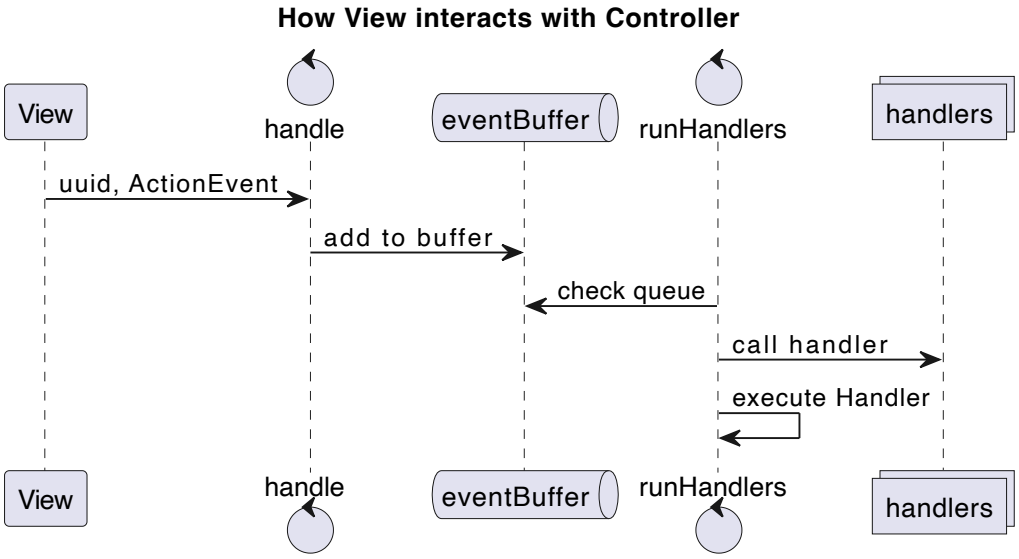
Author: Abir Faisal



Description: A View is registered to a Model. When the model is updated by the Controller the view is notified that the model has changed. From there the view will update itself from the model.

Sequence Diagrams for program continued

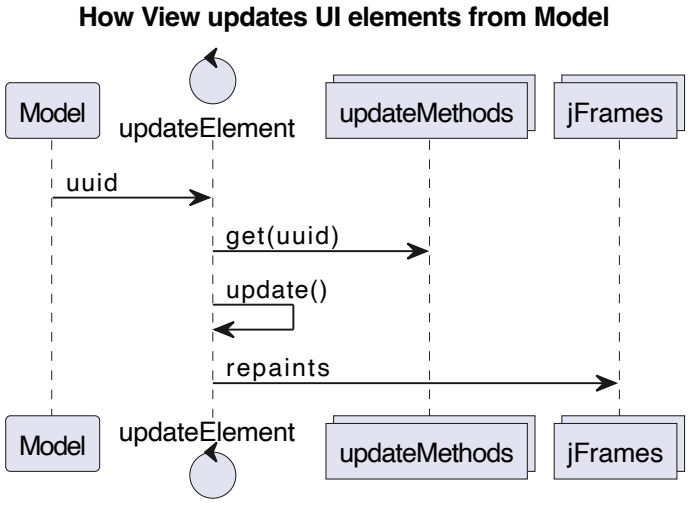
Author: Abir Faisal



Description: When an event occurs in the view the Controller checks the handlers HashMap to see if it contains a corresponding Handler.

If so then the handler is executed.

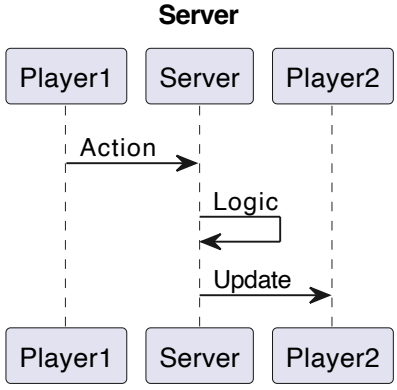
Author: Abir Faisal



Description: Each Swing component has an by declaring an new Updater with the Updater interface and putting it into the updateMethods hashmap.

When the model notifies the view the View calls the corresponding update method executes

Author: Abir Fasial



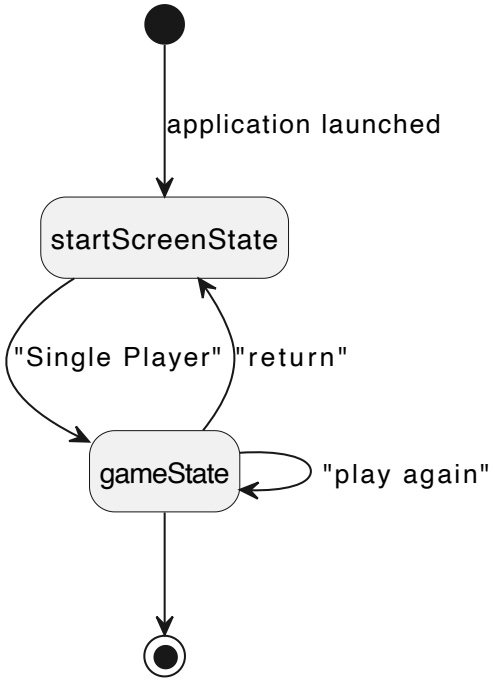
Description: Server We didn't get to this but basically it's a server that would have players connect to it and it would mediate between them.

State Diagrams:

Description: General Application State Diagrams

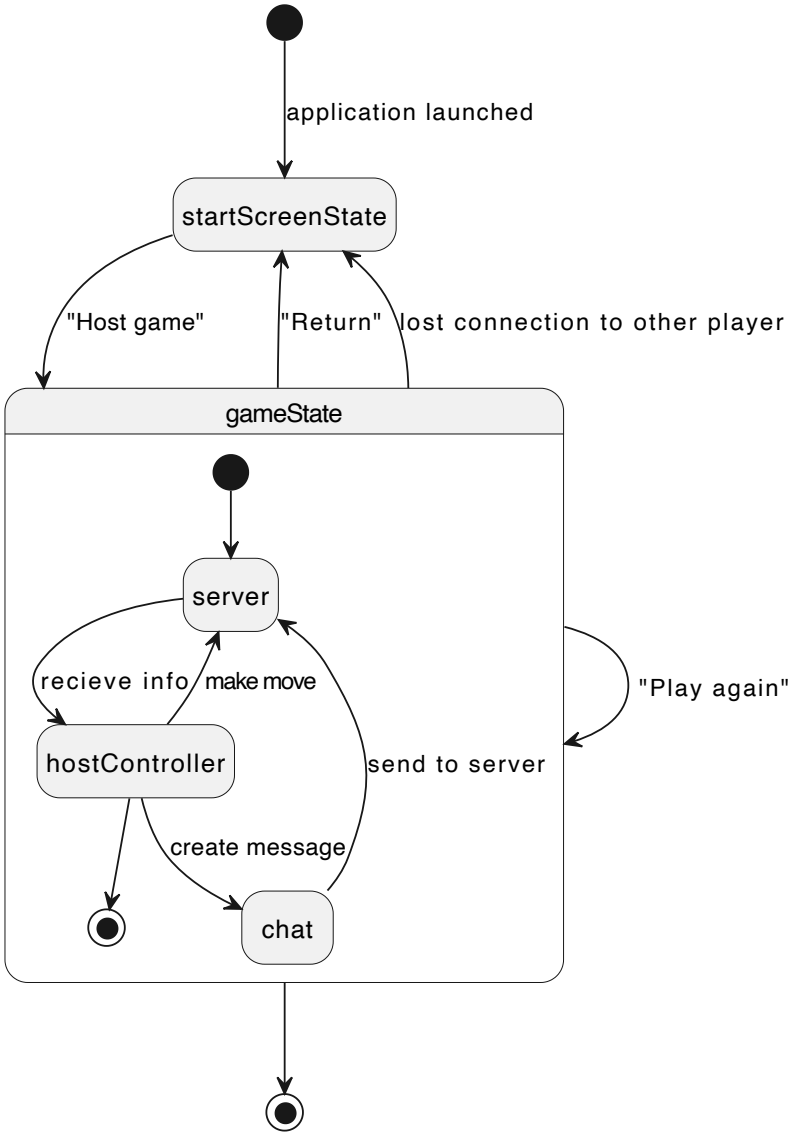
Author: Bryan Barreto

Single Player



Author: Bryan Barreto

Multi Player Host



Author: Bryan Barreto

Multi Player Join