BANG ONLINE (Programmer documentation)

Samuel Jančo

Specification Breakdown

This project is an online version of well-known card game Bang. Users can play online via console application that simulates its card based equivalent.

List and number functional requirements of the project

1. The application allows host create game server.
2. The application allows guests to join the game server.
3. The application allows user to chose a nickname.
4. The application will set up the game of bang (distribute roles, characters and cards).
5. Player on move will always receive two new cards.
6. Player on move can use any of his card/s and the application will execute the effect of chosen card/s.
7. Player affected by effect of another player’s card is able to respond to the effect(missed etc.).
8. Player on move can end his move whenever he wants(if validation passed).
9. The application considers player’s character card when validating actions/reactions.
10. The application validates every player’s action/reaction so it follows game rules.
11. The application switches killed players into spectate mode.
12. The application announces winners at the end of the game.
13. The application reveals (living) players roles at the and of the game.

Architecture / Design

Stages

1. Server initialization (Requirements 1, 2 and 3)
   During this stage user chooses if he’s the host or a guest. In case of being the host he selects how many users will be playing call procedure setting up the server. Then all players send a request to server including their nickname. Server saves the nickname and response with user’s unique ID.

2. Game initialization (Requirement 4)
   During this stage server randomly distributes role and two characters to each player. Players response to server with chosen character and in the end, server distributes cards based on life points of chosen character.

3. Game (Requirements 5,6,7,8,9,10 and 11)
   During this stage players takes turns and making desired moves within the boundaries of rules of the game. Once a player dies his/her role is revealed to others and he can spectate rest of the game.

4. Game ending (Requirements 12 and 13)
   Once the game loop should end the application will finish the game and announce the results containing the winner and role reveals of all players.

Application is split into smaller server part, which consists of server related functions and procedures (set up the server, handling client’s requests, sharing game state json among all players etc.) and bigger client’s/player’s part whose job is practically everything else. It contains methods and procedures to enable to players insert desired moves which are after validation being executed and all players are immediately seeing results of said moves.

Player’s part is managed by multiple objects such as MoveValidator, MoveExecutor, Client, GameMaterial, GameState and UIFormatter.
Each of these classes contains multiple methods and parameters which participates on executing class’s goal.

User input is process via Input class and its methods. Methods validates user’s input and if it has required format it’s return for further use. In case of validation error user is asked to reenter his input.

Technical documentation

List of data structures and their documentation

Game

Takes care of the game from the client’s side

Parameters: client(Client) , gs(GameState), material(GameMaterial), validator(GameValidator), formatter(UIFormatter), rnd(Random), ID(int)

Methods: StartGame, PlayersAliveCount, PlayersAlive, GetSheriffID, GameHasWinner, SaveGameStateJson, GetGameStateJson,GetRole,ChooseCharacter, RestorDeck, GameLoop, HasCard, PlayMove, TranslateMoveString, IdentifyMove

Client

Comunicates with the server.

Parameters: clientSocket(Socket), PORT(int)

Methods: ConnectToServer, SendString, ReciveResponse, Exit

Server

Communicates with all clients and shares GameState json among them. It also distributes roles, characters, and cards during game initialization.

Parameters: serverSocket(Socket), clientsSockets(List<Socket>), BUFFER_SIZE(int), PORT(int), buffer(byte[]), gs(GameState), material(GameMaterial), numberOfPlayers(int), readyPlayers(int)

Methods: SetupServer, DistributeCards, AcceptCallback, SaveGameStateJson, DistributeRolesAndCharacters, GetGameStateJson, SendGameStateJsonToAll, ReceiveCallback, CloseAllSockets

GameState

Stores state of the game. It’s storing info about the game. It stores data about players, used and unused cards in the deck, last event that happened and data about currently ongoing move as well.

Parameters: Players(Dictionary<int,PlayerInfo>), CurrentPlayingID(int), FirstPartDone(bool), PlayedBang(bool), LastEvent(GameEvent), DeckOfCard(DeckOfCards)

Methods:

PlayerInfo

Stores data about player including his ID, name, role, character, life points, card in hand, card on table and alive status.

Parameters: ID(int), Name(string), Role(string), ChosenCharacter(string), Characters(List<string>), Lifes(int), CardOnTable(List<string>), CardInHand(List<string>), ISAlive(bool)

Methods:

DeckOfCard

Stores two Lists of string, the discard pile and the deck of cards.

Parameters: UnusedCard(List<string>), UsedCards(List<string>)

Methods:

GameEvent

Stores an event in the game including ID of the player who
played it, ID of the target of the event and string describing the event.

Parameters: PlayedBy(int), Event(string), PlayedOn(int)

Methods: SetLastEvent

GameMaterial

Stores material used for the game such as Roles, Characters and card strings.

Parameters: rnd(Random), Characters(Dictionary<string, Character>), Roles(Dictionary<string, Role>), BlueCards(List<string>), GunsRanges(Dictionary<string, int>),

Methods: GetShuffledDeck, GetRoles, GetCharacters, DistributeCards

MoveExecutor

Can decide what move is being played and calls the proper function that executes the move.

Parameters: cardList(CardList), material(GameMaterial)

Methods: ExecuteMove, ThrowAwayCardAction, EndMoveAction, ThrowAwayUsedCard, TakeTwoCards, TranslateCardName, HasCard, ChangePlayer

BlueCardsEffectExecutor

Executes effect of blue cards.

Parameters: material(GameMaterial)

Methods: ApplyGunEffect, ApplyScopeEffect, ApplyMustangEffect, ApplyDinamiteEffect, ApplyJailEffect, ApplyBarrelEffect, ApplyVolcanicEffect, HasCard, GetCardValue, TranslateCardName, Die, PassDynamiteTo, ChangePlayer

MoveValidator

Before executing the move, it validates that it doesn’t violate any of the rules.

Parameters: cardList(CardList)

Methods: ValidateMove

InputValidator

Validates user’s input

Parameters:

Methods: ChooseCharacter, ChooseNick, InsertMove

UIFormatter

Formats GameState into friendlier user interface that is displayed to players.

Parameters: material(GameMaterial), divider(string), emptySeat(PlayerInfo)

Methods: SetUpEmptySeat, RenderAll, RenderTable, RenderRowOfPlayers, RenderLives, RenderNameAndLives, RenderRole, RenderBothNamesAndLives, RenderDivider, RenderDivider, RenderCharacters, RenderCardsInHand, RenderCardsOnTable, RenderDiscardPile, RenderLastEvent, RenderSpaces, AlignRows, StringDivider, RenderMyInfo, RenderMyName, RenderMyLives, RenderMyCardsInHand, RenderMyCardsOnTable, RenderMyCharacter, RenderMyRole, RenderResults

Card

Base class for all card types in the game.

Parameters: material(GameMaterial)

Methods: Action, Response, Validate, ChangePlayer, HasCard, TransalteCardName, Die, ThrowAwayUsedCard, UseBlueCardAction, IsInRange, ValidatePossesion, ValidatePossesionAndTarget

CardList

List storing all the card types in the game.

Parameters: Cards(Dictionary<string, Card>)

Methods:

Character

Base class for all character types in the game.

Parameters: Name(string), Ability(string), Lifes(int)

Methods: ApplyAbility, GetCardValue, HasCard

Role

Structure containing data about given role.

Parameters: Name(string), Mission(string)

SomeCharacterType : Character

Character type for character named Bart Cassidy

Parameters:

Methods: ApplyAbility

*Applies for all character types

SomeCardType : Card

Card type for Beer cards

Parameters:

Methods: Action, Validate

*Applies for all cards types

List of procedures and functions and their documentation

Game

public void StartGame()

Connect to the server, present role/character choises and choose one.
Starts game loop.

private int PlayersAliveCount()

Counts how many players are still alive.

private List<PlayerInfo> PlayersAlive()

Returns List of players still alive.

private int GetSheriffID()

Returns ID of the Sheriff.

private string GameHasWinner()

Returns game winner or “None” if there isn’t one so far.

private void SaveGameStateJson()

Gets game state json from the server and saves it.

private string GetGameStateJson()

Serialize game state into json format

private void GetRole()

Presents role of the player

private void ChooseCharacter()

Presents character choises and lets the player chose one.

private void RestoreDeck()

Restrores deck when its empty.

private void GameLoop()

Loop in which players are taking turns to play their moves until the game ends.

private bool HasCard(GameState gs, int player, string card)

Returns whether player possesses a card or not.

private void PlayMove()

Executes players move

private GameEvent IdentifyMove(string moveString)

Translates move string to GameEvent object

Client

public int ConnectToServer()

Connects to server

public void SendString(string text)

Sends a string to server

public string ReceiveResponse()

Receives response from the server

public void Exit()

Disconnects from the server

Server

public void SetupServer()

Sets up the game server

private void DistributeCards()

Distributes cards to all players and deck

private void AcceptCallback(IAsyncResult AR)

Connects client to the server.

private void SaveGameStateJson(string jsonString, string parameter)

Saves given json into GameState object

private void DistributeRolesAndCharacters()

Distributes roles and characters to all players.

private string GetGameStateJson()

Serialize game state into json format.

private void SendGameStateJsonToAll()

Send game state json to all players.

private void ReceiveCallback(IAsyncResult AR)

Receives client's requests.

public void CloseAllSockets()

Disconnects all clients.

public void SetLastEvent(int PlayedBy, string Event, int PlayedOn = -1)

Sets parameters of last event.

GameMaterial

public void GetShuffledDeck(GameState gs)

Shuffles the list of cardstring in the deck.

public void GetRoles(GameState gs)

Randomly chooses role for every player.

public void GetCharacters(GameState gs)

Randomly chooses two characters for every player.

public void DistributeCards(GameState gs)

Distributes the cards for players base on their life points count

MoveExecutor

public void ExecuteMove(GameState gs, GameEvent move)

Executes given move via function or card's method.

private void ThrowAwayCardAction(GameState gs, int playedBy)

Called when player decides to throw away a card. Moves the card from players hand/table to discard pile.

private void EndMoveAction(GameState gs, int playedBy)

Called when player decides to end his move. Changes gs.CurrentPlayerID to next living player.

private void ThrowAwayUsedCard(GameState gs,int playedBy, string cardName)

Called when player uses a card. Moves the card from players hand/table to discard pile.

private void TakeTwoCards(GameState gs, int playedBy)

Called at the beginig of players turn. It adds two cards from deck to his hand.

private string TranslateCardName(GameState gs, int playedBy, string cardNiceName, string handOrTable)

Translates card nice name string to card code name of specific card in player's hand/table.

private bool HasCard(GameState gs, int player, string card, string handOrTable)

Returns whether player possesses a card or not in his hand/ on his table.

private void ChangePlayer(GameState gs)

Changes player to play his moves to next living player.

MoveValidator

public bool ValidateMove(GameState gs, GameEvent move)

Validates given move and return if validation has passed or not.

InputValidator

public int ChooseCharacter()

Validates player's input when choosing character.

public string ChooseNick()

Validates player's input when choosing nick.

public string InsertMove(GameState gs)

Validates player's input when inserting move.

UIFormatter

private void SetUpEmptySeat()

Initializes emptySeat parameter.

public void RenderAll(GameState gs, int myID)

Renders whole game user interface for given player.

private void RenderTable(GameState gs, List<PlayerInfo> playersInfo)

Renders info about other players, deck, and last event.

private void RenderRowOfPlayers((PlayerInfo, PlayerInfo) playerInfo, (int, int) spacing)

Renders info for two players.

private void RenderLives(int Lives)

Renders bullets representing players life points.

private void RenderNameAndLives(string name, int Lives, bool isSheriff, bool isAlive, string role)

Renders name and life points labels for given player. If player’s dead it reveals his role.

private void RenderRole(string role)

Render players role label.

private void RenderBothNamesAndLives((PlayerInfo,PlayerInfo) playersInfo, (int,int) spacing)

Renders line with name and lives labels of two players.

private void RenderDivider((int, int) spacing)

Renders divider.

private void RenderDivider()

Renders divider.

private void RenderCharacters((PlayerInfo, PlayerInfo) playersInfo, (int, int) spacing)

Renders characters label for two players.

private void RenderCardsInHand((PlayerInfo, PlayerInfo) playersInfo, (int, int) spacing)

Renders cards in hand count label.

private void RenderCardsOnTable((PlayerInfo, PlayerInfo) playersInfo, (int, int) spacing)

Renders cards on table label.

private void RenderDiscardPile(string lastCard)

Renders last card in discard pile label.

private void RenderLastEvent(GameState gs)

Renders last event label.

private void RenderSpaces(int amount)

Renders given number of spaces.

private (List<string>, List<string>) AlignRows(List<string> leftPlayer, List<string> rightPlayer)

Aligns lines in List of lines on the same lengths by adding spaces at the end.

private List<string> StringDivider(string stringToDivide, int rowLenght)

Divides longer text into lines with given maximum length.

private void RenderMyInfo(PlayerInfo myInfo)

Renders info about player user is playing for.

private void RenderMyName(string myName, bool isSheriff)

Renders user’s name label.

private void RenderMyLives(int myLives)

Renders bullets to represent user’s life points.

private void RenderMyCardsInHand(PlayerInfo myInfo)

Renders user’s card in hand label.

private void RenderMyCardsOnTable(List<string> myCards)

Renders user’s card on table label.

private void RenderMyCharacter(string myCharacter)

Renders user’s character label.

private void RenderMyRole(string myRole)

Renders user’s role label.

public void RenderResults(GameState gs, string winner)

> Render final results of the game.

Card

public virtual void Action(GameState gs, int playedBy, int playedOn)

> Executes card action effect.

public virtual void Response(GameState gs, int playedBy)

> Execute response on a card.

public virtual bool Validate(GameState gs, GameEvent move)

> Validate card move that player wants to execute.

protected void ChangePlayer(GameState gs)

> Changes player to play his moves to next living player.

protected bool HasCard(GameState gs, int player, string card, string handOrTable)

> Returns whether player possesses a card or not in his hand/ on his table.

protected string TranslateCardName(GameState gs, int playedBy, string cardNiceName, string handOrTable)

> Translates card nice name string to card code name of specific card in player’s hand/table.

protected void Die(GameState gs, int died, int killedBy)

> Kills the player. Switches him to spectate mode and clears all his cards.

protected void ThrowAwayUsedCard(GameState gs, int playedBy, string cardName)

> Called when player uses a card. Moves the card from players hand/table to discard pile.

protected void UseBlueCardAction(GameState gs, int playedBy, string card)

> Executes given blue card action.

protected bool IsInRange(int playedBy, int playedOn, int range, int playersCount)

> Validates if target of the card is in player’s range.

protected bool ValidatePossesion(GameState gs, GameEvent move)

> Validates if player possesses given card.

protected bool ValidatePossesionAndTarget(GameState gs, GameEvent move)

> Validates if player possesses given card and if target of the card is a valid player.

Character

public virtual bool ApplyAbility(GameState gs, int playedBy, int playedOn)

> Verifies if users has given character and if so executes character's ability effect.

protected (int, string) GetCardValue(string cardCodeName)

> Returns cards value.

protected bool HasCard(GameState gs, int player, string card)

> Returns whether player possesses a card or not in his hand.