Московский государственный технический
университет имени Н.Э. Баумана

Парадигмы и конструкции языков программирования
Отчёт по домашнему заданию

Работу выполнил
Студент группы ИУ5-35Б
Шиленок Д.А.

2023 г.

Задание

Создать электронную версию настольной игры “Риичи-маджонг”. Реализовать основные игровые механики в соответствии с правилами игры, а именно:

- Взятие и сброс тайлов (специальных прямоугольных костей с различными рисунками, с помощью которых ведется игра) в дискард (область на игровом поле, где хранятся сброшенные игроками тайлы)
- Объявление победы: по цумо (взятие последнего нужного для победы тайла) и по рон (взятие последнего нужного для победы тайла у бота). А также рассмотрение ситуаций, когда игрок не победил по рон или цумо в течение одного раунда.
- Смена раунда и подсчет очков
- Окончание игры

Текст программы

Wall.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Riichi_mahjong
{
   class Wall
   {
      const int MAX_DORA_COUNT = 5;
      const int TILES_IN_DWALL = 14;
      int doraCount;
      Stack<Tile> wall = new Stack<Tile>();
      List<Tile> deadWall = new List<Tile>();
      public event EventHandler? TilesEnded;

      ITileRandomiser randomiser;
      public Wall(ITileRandomiser randomiser)
      {
         this.randomiser = randomiser;
         Reset();
      }

      public void Reset()
      {
         doraCount = 1;
         randomiser.Reset();
         wall.Clear();
         deadWall.Clear();
         while (deadWall.Count < TILES_IN_DWALL && randomiser.Sum() > 0)
deadWall.Add(randomiser.GetTile());
         while (randomiser.Sum() > 0) wall.Push(randomiser.GetTile());
      }

      public Tile GetTileFromWall()
      {
         if (TilesLeft() > 0) return wall.Pop();
         else TilesEnded?.Invoke(this, EventArgs.Empty);
         return Tile.m1;
      }

      public bool OpenDora()
      {
         if (doraCount < MAX_DORA_COUNT)
         {
            doraCount++;
```

```
            return true;
        }
        return false;
    }

    public List<Tile> GetDoraList()
    {
        List<Tile> doraList = new List<Tile>();
        for (int i = 0; i < doraCount; i++) doraList.Add(deadWall[i]);
        return doraList;
    }

    public List<Tile> GetUraDoraList()
    {
        List<Tile> uraDoraList = new List<Tile>();
        for (int i = 0; i < doraCount; i++) uraDoraList.Add(deadWall[i + TILES_IN_DWALL / 2]);
        return uraDoraList;
    }

    public override string ToString()
    {
        string str = string.Empty;
        foreach (Tile tile in wall) { str += tile.ToString() + "\n"; }
        str += "\n";
        foreach (Tile tile in deadWall) { str += tile.ToString() + " "; }
        return str;
    }
    public int TilesLeft() => wall.Count;
  }
}
```

TileRandomiser.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;
using System.Reflection.Metadata.Ecma335;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;

namespace Riichi_mahjong
{
  interface ITileRandomiser
  {
    public void Reset();
    public Tile GetTile();
    public int Sum();
  }
  class TileRandomiser : ITileRandomiser
  {
```

```
        Random rand;
        int[] tiles;
        HashSet<int> exclude;
        public TileRandomiser()
        {
            rand = new Random();
            tiles = new int[(int)Tile.last - 1];
            exclude = new HashSet<int>();
            Reset();
        }
        public void Reset()
        {
            for (int i = 0; i < tiles.Length; i++)
            {
                if (i != (int)Tile.m5A && i != (int)Tile.p5A && i != (int)Tile.s5A) tiles[i] = 4;
                else tiles[i] = 1;
            }
            exclude.Clear();
        }
        public Tile GetTile()
        {
            var tile = new Tile();
            var range = Enumerable.Range(0, (int)Tile.last - 1).Where(i => !exclude.Contains(i));
            int index = rand.Next(0, (int)Tile.last - 1 - exclude.Count);
            tile = (Tile)range.ElementAt(index);
            if (--tiles[(int)tile] == 0) exclude.Add((int)tile);
            return tile;
        }

        public int Sum()
        {
            return tiles.Sum();
        }
    }
}
```

Tile.cs

```
namespace Riichi_mahjong
{
    enum Tile
    {
        m1, m2, m3, m4, m5, m5A, m6, m7, m8, m9,
        p1, p2, p3, p4, p5, p5A, p6, p7, p8, p9,
        s1, s2, s3, s4, s5, s5A, s6, s7, s8, s9,
        red, white, green,
        east, south, west, north,
        last
    }
}
```

Player.cs

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Riichi_mahjong
{
   enum Wind
   { east, south, west, north }
   class Player
   {
      const int DEFAULT_SCORE = 25000;
      const int RIICHI_COST = 1000;
      private const int TILES_IN_HAND = 13;

      public Hand Hand { get; }
      public List<Tile> Discard { get; }
      public int Score { get; private set; }
      public bool IsDealer { get; private set; }
      public Wind SeatWind { get; private set; }
      public Player(Wind seatWind)
      {
         Hand = new Hand();
         Discard = new List<Tile>();
         Score = DEFAULT_SCORE;
         SeatWind = seatWind;
         if (seatWind == Wind.east) { IsDealer = true; }
      }

      public Tile CallRiichi(int index)
      {
         if (Score > RIICHI_COST)
         {
            Score -= 1000;
            return DropTile(index);
         }
         else throw new Exception("Not enough points to call a riichi");
      }
      public virtual Tile DropTile(int index)
      {
         Tile droppedTile = Hand.DropTile(index);
         Discard.Add(droppedTile);
         return droppedTile;
      }
      public void PickTile(Wall wall)
      {
         Hand.PutTile(wall.GetTileFromWall());
      }
```

```
        public void NextWind()
        {
            SeatWind += (int)SeatWind < 3 ? 1 : -3;
            IsDealer = SeatWind == Wind.east;
        }
        public void AddScore(int score)
        {
            Score += score;
        }
        public void FillHand(Wall wall)
        {
            for(int i=0;i<TILES_IN_HAND;i++)
            {
                PickTile(wall);
            }
        }
        public void SortHand()
        {
            Hand.SortHand();
        }
    }

    class Bot : Player
    {
        public Bot(Wind seatWind) : base(seatWind) { }

        public override Tile DropTile(int index)
        {
            return base.DropTile(0);
        }
    }
}
```

Hand.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Riichi_mahjong
{
    class Hand
    {
        public List<Tile> Tiles { get; }
        public Hand()
        {
            Tiles = new List<Tile>();
        }
        public void PutTile(Tile tile)
        {
```

```
            Tiles.Add(tile);
        }
        public Tile DropTile(int index)
        {
            Tile tile = Tiles[index];
            Tiles.RemoveAt(index);
            return tile;
        }
        public void SortHand()
        {
            Tiles.Sort();
        }
    }
}
```

GameTime.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Riichi_mahjong
{
    class GameTime
    {
        public float TimeScale { get; set; }
        public float DeltaTime
        {
            get { return TimeScale * DeltaTime; }
            set { }
        }
        public float TotalTimeElapsed { get; private set; }
        public void Update(float deltaTime, float totalTimeElapsed)
        {
            DeltaTime = deltaTime;
            TotalTimeElapsed = totalTimeElapsed;
        }
    }
}
```

GameLoop.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SFML.Graphics;
using SFML.Window;
using SFML.System;
```

```
namespace Riichi_mahjong
{
   abstract class GameLoop
   {
      public const int TARGET_FPS = 60;
      public const float TIME_UNTIL_UPDATE = 1f / TARGET_FPS;

      public RenderWindow Window { get; protected set; }
      public GameTime GameTime { get; protected set; }
      public Color WindowClearColor { get; protected set; }
      protected GameLoop(uint windowWidth, uint windowHeight, string windowTitle, Color
windowClearColor)
      {
         this.WindowClearColor = windowClearColor;
         this.Window = new RenderWindow(new VideoMode(windowWidth, windowHeight),
windowTitle);
         this.GameTime = new GameTime();
         Window.Closed += WindowClosed;
      }

      private void WindowClosed(object? sender, EventArgs e)
      {
         Window.Close();
      }

      public void Run()
      {
         LoadContent();
         Initialize();

         float totalTimeBeforeUpdate = 0;
         float previousTimeElapsed = 0;
         float deltaTime = 0;
         float totalTimeElapsed = 0;

         Clock clock = new Clock();

         while(Window.IsOpen)
         {
            Window.DispatchEvents();
            totalTimeElapsed = clock.ElapsedTime.AsSeconds();
            deltaTime = totalTimeElapsed - previousTimeElapsed;
            previousTimeElapsed = totalTimeElapsed;

            totalTimeBeforeUpdate += deltaTime;

            if (totalTimeBeforeUpdate >= TIME_UNTIL_UPDATE)
            {
               GameTime.Update(totalTimeBeforeUpdate, clock.ElapsedTime.AsSeconds());
               totalTimeBeforeUpdate = 0;
               Update(GameTime);
```

```
            Window.Clear(WindowClearColor);
            Draw(GameTime);
            Window.Display();
          }
        }
      }

      public abstract void LoadContent();
      public abstract void Initialize();
      public abstract void Update(GameTime gameTime);
      public abstract void Draw(GameTime gameTime);
    }
}
```

Discard.cs

```
using SFML.Graphics;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Riichi_mahjong
{
  internal class Discard : Drawable
  {
    TileTextures tileTextures;
    public List<Tile> DiscardList { get; private set; }
    public List<Sprite> TileSprites { get; set; }
    public Discard(Player player, TileTextures tileTextures)
    {
      DiscardList = player.Discard;
      TileSprites = new List<Sprite>();
      this.tileTextures = tileTextures;
    }

    public void Update()
    {
      int k = 0;
      TileSprites.Clear();
      foreach (Tile tile in DiscardList)
      {
        Sprite sprite = new Sprite(new Texture(tileTextures.GetTileTexture(tile)));
        int X = (int)(Positions.PlayerDiscardPosition.X + (tileTextures.Textures[Tile.m1].Size.X + 10) *
(k % 6));
        int Y = (int)(Positions.PlayerDiscardPosition.Y + (tileTextures.Textures[Tile.m1].Size.Y + 5) *
(k / 6));
        sprite.Position = new SFML.System.Vector2f(X, Y);
        TileSprites.Add(sprite);
        k++;
      }
```

```
        }

        public void Draw(RenderTarget target, RenderStates states)
        {
            foreach(Sprite sprite in TileSprites)
            {
                sprite.Draw(target, states);
            }
        }
    }
}
```

Positions.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SFML.System;

namespace Riichi_mahjong
{
    static class Positions
    {
        public static readonly Vector2f HandPosition = new Vector2f(225, 900);
        public static readonly Vector2f PlayerDiscardPosition = new Vector2f(375, 650);
        public static readonly Vector2f LeftBotDiscardPosition = new Vector2f(10, 20);
        public static readonly Vector2f RightBotDiscardPosition = new Vector2f(20, 20);
        public static readonly Vector2f UpBotDiscardPosition = new Vector2f(30, 20);
        public static readonly Vector2f scoreboardPosition = new Vector2f(0, 0);
        public static readonly Vector2f RiichiButtonPosition = new Vector2f(0, 0);
    }
}
```

TileTextures.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SFML.Graphics;
using SFML.System;

namespace Riichi_mahjong
{
    class TileTextures
    {
        public Dictionary<Tile,Texture> Textures { get; private set; }
        public TileTextures(string TexturesFolderName)
        {
            Textures = new Dictionary<Tile,Texture>();
            for(Tile tile = 0;tile < Tile.last; tile++)
```

```
        {
            Textures.Add(tile, new Texture(TexturesFolderName + tile + ".png"));
        }
    }

    public Texture GetTileTexture(Tile tile)
    {
        return Textures[tile];
    }
    public Vector2f GetDimensions()
    {
        return new Vector2f(Textures[Tile.m1].Size.X, Textures[Tile.m1].Size.Y);
    }
  }
}
```

Game.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SFML.Window;
using SFML.System;
using SFML.Graphics;
using SFML.Audio;


namespace Riichi_mahjong
{
  enum GameState
  {
    NewRound,
    PlayerPick,
    PlayerMove,
    BotMove,
    End
  }
  class Game: GameLoop
  {
    public const uint WINDOW_WIDTH = 1000;
    public const uint WINDOW_HEIGHT = 1000;
    public const string TEXTURES_FOLDER_NAME = "Textures/";
    public GameState State {  get; set; }
    public bool IsLastTile { get; set; }
    public Sprite? Background { get; set; }
    public Texture? BackgroundTexture { get; set; }
    public TileTextures? TileTextures { get; set; }
    public Wall Wall { get; set; }
    public Player Player { get; set; }
    public List<Bot> Bots { get; set; }
```

```
public UI? UI { get; set; }
public Discard? Discard { get; set; }
public Game() : base(WINDOW_HEIGHT, WINDOW_HEIGHT, "RIICHI MAHJONG", Color.Black)
{
   Wall = new Wall(new TileRandomiser());
   State = GameState.PlayerPick;
   IsLastTile = false;
   Wall.TilesEnded += TilesEnded;
   Player = new Player(Wind.east);
   Player.FillHand(Wall);
   Player.SortHand();
   Bots = new List<Bot>();
   Window.MouseButtonPressed += PlayerMove;
}

private void PlayerMove(object? sender, MouseButtonEventArgs e)
{
   int index = UI.CheckTileClick(e);
   if (State == GameState.PlayerMove)
   {
      if (index != -1) {
         Player.DropTile((int)index);
         Player.SortHand();
         if (IsLastTile) State = GameState.End;
         else State = GameState.BotMove;
      }
   }
}

private void TilesEnded(object? sender, EventArgs e)
{
   IsLastTile = true;
}

public override void Draw(GameTime gameTime)
{
   Window.Draw(Background);
   Window.Draw(UI);
   Window.Draw(Discard);
}

public override void Initialize()
{
   Bots.Add(new Bot(Wind.south));
   Bots[0].FillHand(Wall);
   Bots.Add(new Bot(Wind.west));
   Bots[1].FillHand(Wall);
   Bots.Add(new Bot(Wind.north));
   Bots[2].FillHand(Wall);
}
```

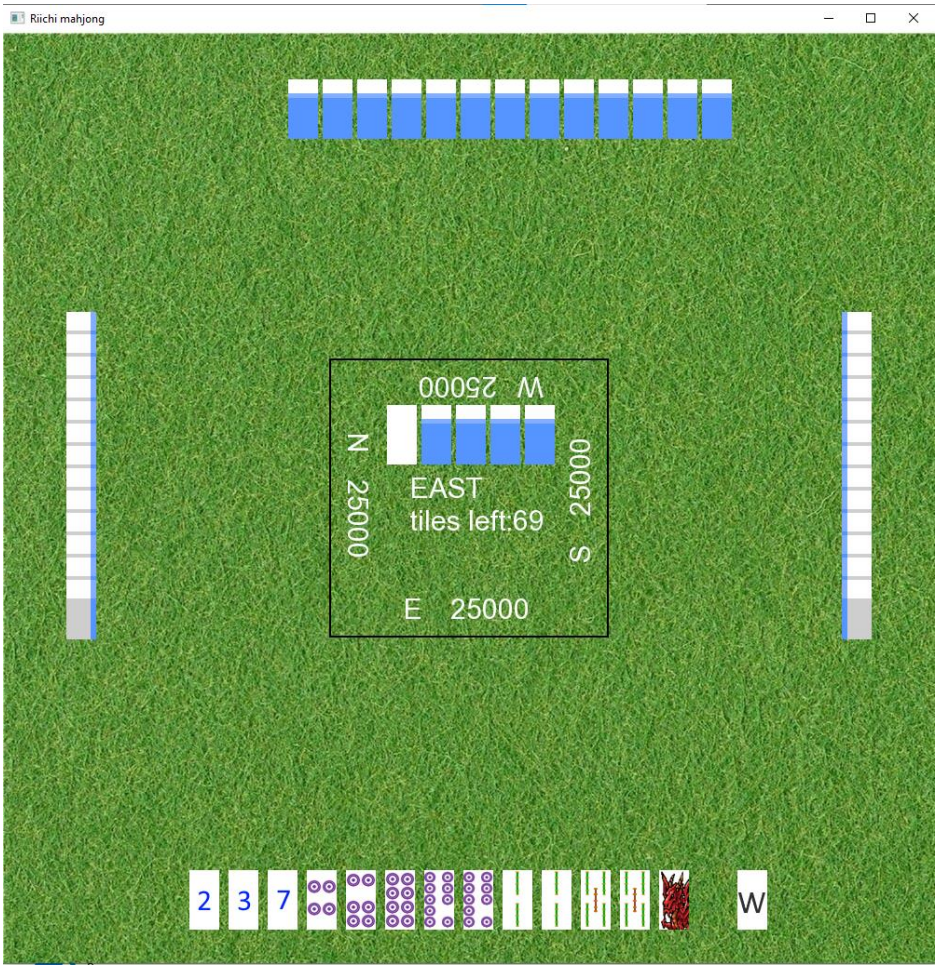
```
        public override void LoadContent()
        {
            BackgroundTexture = new Texture("Textures/bg.png");
            Background = new Sprite(BackgroundTexture, new IntRect(0, 0, (int)WINDOW_WIDTH,
(int)WINDOW_HEIGHT));
            TileTextures = new TileTextures(TEXTURES_FOLDER_NAME);
            UI = new UI(Player, TileTextures);
            Discard = new Discard(Player, TileTextures);
        }

        public override void Update(GameTime gameTime)
        {
            switch (State)
            {
                case GameState.PlayerPick:
                    Player.PickTile(Wall);
                    State = GameState.PlayerMove;
                    break;
                case GameState.BotMove:
                    foreach (Bot bot in Bots)
                    {
                        bot.PickTile(Wall);
                        if (IsLastTile)
                        {
                            bot.DropTile(0);
                            State = GameState.End;
                            break;
                        }
                    }
                    if (State != GameState.End) State = GameState.PlayerPick;
                    break;
                case GameState.End:
                    Console.WriteLine("END");
                    break;
            }
            Discard?.Update();
            UI?.Update();
        }
    }
}
```

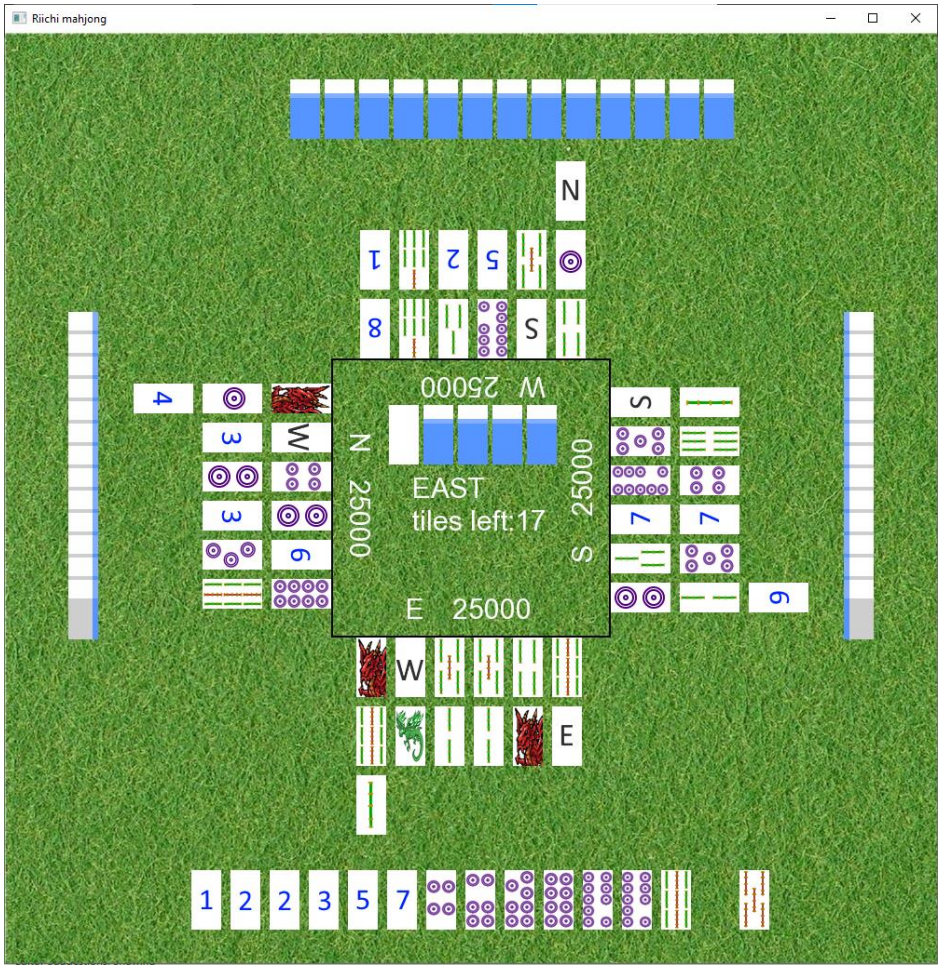
Program.cs

```
using SFML.Graphics;
using SFML.Window;

namespace Riichi_mahjong
{
    class Program
    {

        static void Main(string[] args)
```

```
        {
            Game game = new Game();
            game.Run();
        }
    }
}
```

Скриншоты программы




Riichi mahjong
N
1
2
5
8
S
W 25000
4
3
W
3
6
N 25000
EAST
tiles left:17
S 25000
S
7
7
9
E 25000
W
E
1 2 2 3 5 7