Programmer's Guide, CS 4500

Latin Squares

Group 7

Amal Presingu (anpkcd@umsystem.edu)

Omar Shabazz (omsm94@mail.umsl.edu)

Tan Nguyen (ntnhmc@mail.umsl.edu)

Caleb Kimberlin (cmkd7h@mail.umsl.edu)

Jude Ohene Boateng (jjoheneboateng@mail.umsl.edu)

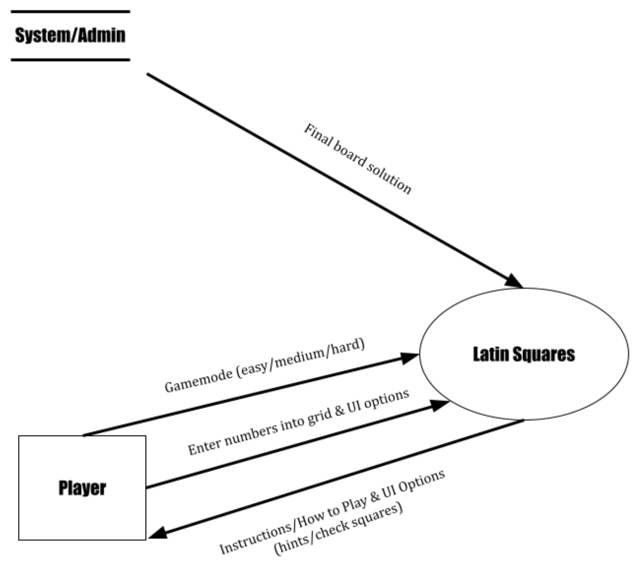
First Revision: 4/12/2022

Second Revision: 5/4/2022

Assumptions about the Programmer

Maintenance and installation will not be necessary for our game. The project was developed through Unity, a game engine that allows flexibility and implementation through browsers. Unity scripting is done through C#, a language very similar to Java. We did not have the luxury of version control when working with Unity, so we manually updated files through a dropbox that changed once someone added something new. Although it was tedious, it was the most efficient way of collaboration in our small group.

High Level Design


System/Admin
Final board solution
Latin Squares
Gamemode (easy/medium/hard)
Enter numbers into grid & UI options
Player
Instructions/How to Play & UI Options (hints/check squares)

Amal Presingu (Team Leader)

Omar Shabazz

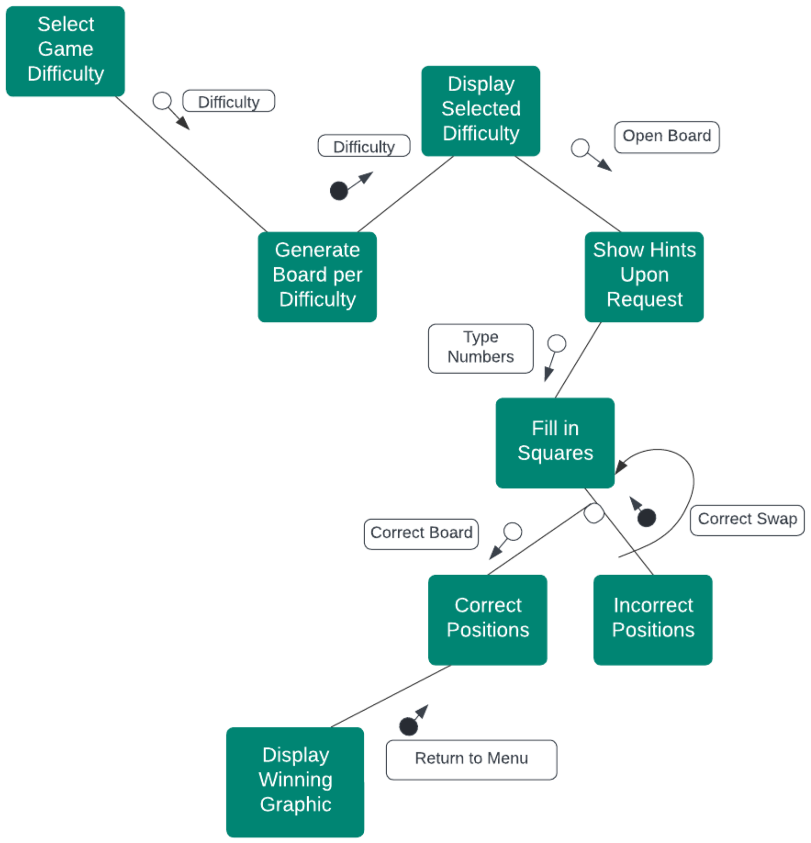
Tan Nguyen

Caleb Kimberlin

Jude Ohene Boateng

3/9/2022

Structure Chart



Pseudocode

```
FUNCTION main
    DEFINE difficulty
    READ user input
    STORE difficulty = user input
    DEFINE struct deck[4][13]
    DEFINE struct selection[difficulty][difficulty]
    DEFINE string name
    WHILE name != string
        PRINT “Enter Username”
        READ user input
        STORE name = user input
    END WHILE

    PRINT “How to play message”

    CALL createDeck
    CALL createSelection

    DEFINE SET status = false
    WHILE check = false
        CALL printSelection
        CLICK CARD user input
        STORE row and column
        CALL swap with selection[row][column],selection[row][column]
        SET status = CALL check RETURNING status
    END WHILE

    PRINT “Winning message”
END FUNCTION
//////////////////////////////////////////////////////////////
FUNCTION createDeck
    DECLARE i = 0,ii = 0
    FOR(i = 0; i < 4; i++)
        FOR(ii = 0; ii < 13; ii++)
            deck[i][ii].suit = i;
            deck[i][ii].rank = ii;
        END FOR
    END FOR
END FUNCTION
//////////////////////////////////////////////////////////////
FUNCTION shuffle
    FOR(int i = 0; i < 4; i++)
        DEFINE int r = rand() % 4
        FOR(int j = 0; j < 13; j++)
            DEFINE SET rr = rand() % 13
            DEFINE SET struct temp = deck[i][j]
            SET deck[i][j] =  deck[r][rr]
            SET deck[r][rr] = temp
        END FOR
        END FOR
END FUNCTION
////////////////////////////////////////////////////////////////////
FUNCTION createSelection
        DEFINE int j = 0,jj = 0,i = 0,ii = 0
        CALL shuffle

        FOR(i = 0; i < difficulty; i++)
                FOR(ii = 0; ii < difficulty; ii++)
                IF jj > 12
                        SET jj = 0
                        SET j = j + 1
                END IF
                SET selection[i][ii] = deck[j][jj]
                SET jj = jj + 1
                END FOR
        END FOR
END FUNCTION
////////////////////////////////////////////////////////////////////////////////
FUNCTION printSelection
        FOR(int row = 0; row < selectionNum; row++)
                FOR(int col = 0; col < selectionNum; col++)
                        PRINT "selection[row][col].suit,selection[row][col].rank"
                END FOR
        END FOR
END FUNCTION
///////////////////////////////////////////////////////////////////////////////
FUNCTION swap struct *a,struct *b
        DEFINE struct t
        SET t = *a;
        SET *a = *b;
        SET *b = t;
END FUNCTION
///////////////////////////////////////////////////////////////////////////////////
FUNCTION check
        DEFINE status = true
        DEFINE int i = 0
        DEFINE row =  < Integer >
        DEFINE col = < Integer >
        FOR(i = 0; i < difficulty && status; i++)
        FOR(int j = 0; j < c && status; j++)
            IF selection[j][i] > difficulty || selection[j][i] < 1 || selection[i][j] >
difficulty  || selection[i][j] < 1
                SET status = false
            ELSE
                SET row add to selection[j][i]
                SET col add to selection[i][j]
            END IF
        END FOR
        IF status && row.size() != difficulty  || col.size() != difficulty
            SET status = false
        END IF
    SET clear row
    SET clear col
    END FOR
    RETURN status
END FUNCTION
```

<u>**Installation Instructions**</u>

To run the program, simply click on the given link. The game will be embedded into a window that the user can fullscreen. There are no required permissions, and the files don't need to be saved anywhere in order to run/test the program. If the user wants to view or download the file contents, navigate to the user manual for the specified links.

APPENDIX A

IMPLEMENTATION CODE

```
//BOARD CODE
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Board : MonoBehaviour
{
   int[,] solvedGrid = new int[9, 9];
   string s;

   int[,] riddleGrid = new int[9, 9];
   int piecesToErase = 35;

   public Transform A1, A2, A3, B1, B2, B3, C1, C2, C3;
   public GameObject buttonPrefab;

   public AudioSource winSound;
   public AudioSource wrongSound;
   public AudioSource nohintSound;

   List<NumberField> fieldList = new List<NumberField>();

   public enum Difficulties
   {
      DEBUG,
      EASY,
      MEDIUM,
      HARD
   }

   public GameObject winPanel;

   public Difficulties difficulty;

   int maxHints;

   void Start()
   {
      winPanel.SetActive(false);

      difficulty = (Board.Difficulties)Settings.difficulty;

      InitGrid(ref solvedGrid);

      ShuffleGrid(ref solvedGrid, 5);
      CreateRiddleGrid();

      CreateButtons();
   }

   void InitGrid(ref int[,] grid)
   {
      for (int i = 0; i < 9; i++)
      {
          for (int j = 0; j < 9; j++)
          {
              grid[i, j] = (i * 3 + i / 3 + j) % 9 + 1;
          }
      }
   }

   void DebugGrid(ref int[,] grid)
   {
      s = "";
      int sep = 0;
      for (int i = 0; i < 9; i++)
      {
          s += "|";
          for (int j = 0; j < 9; j++)
          {
              s += grid[i, j].ToString();

              sep = j % 3;
              if (sep == 2)
              {
                  s += "|";
              }
          }
          s += "\n";
      }
   }

   void ShuffleGrid(ref int[,] grid, int shuffleAmount)
   {
      for (int i = 0; i < 9; i++)
      {
          int value1 = Random.Range(1, 10);
          int value2 = Random.Range(1, 10);
          MixTwoGridCells(ref grid, value1, value2);
      }
   }

   void MixTwoGridCells(ref int[,] grid, int value1, int value2)
   {
      int x1 = 0;
      int x2 = 0;
      int y1 = 0;
      int y2 = 0;

      for (int i = 0; i < 9; i += 3)
      {
          for (int k = 0; k < 9; k += 3)
          {
              for (int j = 0; j < 3; j++)
              {
                  for (int l = 0; l < 3; l++)
                  {
                      if (grid[i + j, k + l] == value1)
                      {
                          x1 = i + j;
                          y1 = k + l;
                      }

                      if (grid[i + j, k + l] == value2)
                      {
                          x2 = i + j;
                          y2 = k + l;
                      }
                  }
              }
              grid[x1, y1] = value2;
              grid[x2, y2] = value1;
          }
      }
   }

   void CreateRiddleGrid()
   {
      for (int i = 0; i < 9; i++)
      {
          for (int j = 0; j < 9; j++)
          {
              riddleGrid[i, j] = solvedGrid[i, j];
          }
      }

      SetDifficulty();


      for (int i = 0; i < piecesToErase; i++)
      {
          int x1 = Random.Range(0, 9);
          int y1 = Random.Range(0, 9);

          while(riddleGrid[x1, y1] == 0)
          {
              x1 = Random.Range(0, 9);
              y1 = Random.Range(0, 9);
          }

          riddleGrid[x1, y1] = 0;
      }

   }

   void CreateButtons()
   {
      for (int i = 0; i < 9; i++)
      {
          for (int j = 0; j < 9; j++)
          {
              GameObject newButton = Instantiate(buttonPrefab);

              NumberField numField = newButton.GetComponent<NumberField>();
              numField.SetValues(i, j, riddleGrid[i, j], i + "," + j, this);
              newButton.name = i + "," + j;

              if(riddleGrid[i, j] == 0)
              {
                  fieldList.Add(numField);
              }

              //A1
              if (i < 3 && j < 3)
              {
                  newButton.transform.SetParent(A1, false);
              }
              //A2
              if (i < 3 && j > 2 && j < 6)
              {
                  newButton.transform.SetParent(A2, false);
              }
              //A3
              if (i < 3 && j > 5)
              {
                  newButton.transform.SetParent(A3, false);
              }
              //B1
              if (i > 2 && i < 6 && j < 3)
              {
                  newButton.transform.SetParent(B1, false);
              }
              //B2
              if (i > 2 && i < 6 && j > 2 && j < 6)
              {
                  newButton.transform.SetParent(B2, false);
              }
              //B3
              if (i > 2 && i < 6 && j > 5)
              {
                  newButton.transform.SetParent(B3, false);
              }
              //C1
              if (i > 5 && j < 3)
              {
                  newButton.transform.SetParent(C1, false);
              }
              //C2
              if (i > 5 && j > 2 && j < 6)
              {
                  newButton.transform.SetParent(C2, false);
              }
              //C3
              if (i > 5 && j > 5)
              {
                  newButton.transform.SetParent(C3, false);
              }

          }
      }
   }
```

```
public void SetInputInRiddleGrid(int x, int y, int value)
{
   riddleGrid[x, y] = value;
}

void SetDifficulty()
{
   switch(difficulty)
   {
      case Difficulties.DEBUG:
         piecesToErase = 5;
         maxHints = 3;
         break;
      case Difficulties.EASY:
         piecesToErase = 30;
         maxHints = 5;
         break;
      case Difficulties.MEDIUM:
         piecesToErase = 40;
         maxHints = 3;
         break;
      case Difficulties.HARD:
         piecesToErase = 60;
         maxHints = 2;
         break;
   }
}

public void CheckComplete()
{
   if (winCondition())
   {
      winSound.Play();
      winPanel.SetActive(true);
   }
   else
   {
      wrongSound.Play();
   }
}

bool winCondition()
{
   for (int i = 0; i < 9; i++)
   {
      for (int j = 0; j < 9; j++)
```

```
            {
                if (riddleGrid[i, j] != solvedGrid[i, j])
                {
                    return false;
                }
            }
        }
        return true;
    }

    public void ShowHint()
    {
        if (fieldList.Count > 0 && maxHints > 0)
        {
            int randIndex = Random.Range(0, fieldList.Count);

            maxHints--;
            riddleGrid[fieldList[randIndex].GetX(), fieldList[randIndex].GetY()] =
solvedGrid[fieldList[randIndex].GetX(), fieldList[randIndex].GetY()];


            fieldList[randIndex].SetHint(riddleGrid[fieldList[randIndex].GetX(),
fieldList[randIndex].GetY()]);
            fieldList.RemoveAt(randIndex);
        }
        else
        {
            nohintSound.Play();
        }
    }
}
```

```
//BUTTON SETTINGS SCRIPT
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class ButtonSettings : MonoBehaviour
{
    public void ButtonClick(string setting)
    {
        if (setting == "easy")
        {
            Settings.difficulty = Settings.Difficulties.EASY;
        }

        if (setting == "medium")
        {
            Settings.difficulty = Settings.Difficulties.MEDIUM;
        }

        if (setting == "hard")
        {
            Settings.difficulty = Settings.Difficulties.HARD;
        }

        SceneManager.LoadScene("SampleScene");
    }

    public void PlayAgain()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }

    public void ReturnToMenu()
    {
        SceneManager.LoadScene("Main Menu");
    }
}
```

```
//INPUT FIELD SCRIPT
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class InputField : MonoBehaviour
{
    public static InputField instance;

    NumberField lastField;

    void Awake()
    {
        instance = this;
    }
    void Start()
    {
        this.gameObject.SetActive(false);
    }

    public void ActivateInputField(NumberField field)
    {
        this.gameObject.SetActive(true);
        lastField = field;
    }

    public void ClickedInput(int number)
    {
        lastField.ReceiveInpout(number);
        this.gameObject.SetActive(false);
    }
}
```

```
//NUMBER FIELD SCRIPT
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class NumberField : MonoBehaviour
{
    Board board;

    int x1, y1;
    int value;

    string identifier;

    public Text number;

    public void SetValues(int _x1, int _y1, int _value, string _identifier, Board _board)
    {
        x1 = _x1;
        y1 = _y1;
        value = _value;
        identifier = _identifier;
        board = _board;

        number.text = (value != 0) ? value.ToString() : "";

        if (value != 0)
        {
            GetComponentInParent<Button>().interactable = false;
        }
        else
        {
            number.color = Color.red;
        }
    }

    public void ButtonClick()
    {
        InputField.instance.ActivateInputField(this);
    }

    public void ReceiveInpout(int newValue)
    {
        value = newValue;
        number.text = (value != 0) ? value.ToString() : "";

        board.SetInputInRiddleGrid(x1, y1, value);
    }

    public int GetX()
    {
        return x1;
    }

    public int GetY()
    {
        return y1;
    }

    public void SetHint(int _value)
    {
        value = _value;
        number.text = value.ToString();
        number.color = Color.green;
        GetComponentInParent<Button>().interactable = false;
    }
}
```

```
//SETTINGS SCRIPT
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Settings : MonoBehaviour
{
   public enum Difficulties
   {
      DEBUG,
      EASY,
      MEDIUM,
      HARD
   }

   public static Difficulties difficulty;
}
```

APPENDIX B

TEST PLAN

Introduction

Our finished product is meant to help students improve their logical thinking skills. The Latin square is the basis of many different puzzles and games. Sudoku is one of the more popular forms of the Latin square. Students may start off a bit confused, but once they solve their first puzzle, they will slowly see improvement. The main technique with Sudoku and other Latin squares is the process of elimination. Within each row, colum, and 3x3 square, there is only a single combination that works. In other words, there is only one solution for each board.

Items to be Tested

- **2.1 - Latin Square Program**: This unit is to denote the functionality of the program itself, which will be tested by a user/tester clicking on the website link and determining whether the program successfully opens/displays the correct information on the starting screen. If the program loads in the browser without crashing and displays the main menu, then the test is a success.

Test 1: 4/22/22, Caleb Kimberlin. Passed

- **2.2 - Difficulty Button:** This unit is to denote the functionality of the difficulty buttons, which will be tested by a user/tester clicking on one of the three difficulty buttons, and determining whether the program successfully launches a game of Latin Squares with the desired difficulty in mind. If the program creates a game of Latin Squares that accurately reflects the chosen difficulty without crashing, then the test is a success.

  Test 1 (Easy): 4/22/22, Caleb Kimberlin. Passed

  Test 2 (Medium): 4/22/22, Caleb Kimberlin. Passed

  Test 3 (Hard): 4/22/22, Caleb Kimberlin. Passed

- **2.3 - Hint Button:** This unit is to denote the functionality of the "Hint" button, which will be tested by a user/tester clicking on the "Hint" button during a Latin Square game, and determining whether the program successfully filled in an empty space for them. If the program fills in an empty square with the correct value for that square after the user hits the button without crashing the program or going over the limit set by the chosen difficulty, then the test is a success.

  Test 1: 4/22/22, Caleb Kimberlin, Passed on Easy difficulty

  Test 2: 4/22/22, Caleb Kimberlin, Passed on Medium difficulty

  Test 3: 4/22/22, Caleb Kimberlin, Passed on Hard difficulty

- **2.4 - Check Squares Button:** This unit is to denote the functionality of the "Check Squares" button, which will be tested by a user/tester clicking on the "Check Squares" button during and at the end of a Latin Squares game, and determining whether the game ends or makes the user continue to play the game. If the program does not display the game over screen when the user still has incorrect/empty spaces on the board upon clicking the button, and the program displays the game over screen when the user has successfully filled in all spaces on the board correctly upon clicking the button without crashing the program, then the test is a success.

Test 1: 4/22/22, Caleb Kimberlin, Passed on Easy difficulty

Test 2: 5/5/22, Amal Presingu, Passed on Medium difficulty

Test 3: 5/5/22, Amal Presingu, Passed on Hard difficulty

- **2.5 - Unfilled Squares:** This unit is to denote the functionality of the unfilled squares within each game of Latin Squares, which will be tested by a user/tester clicking on an empty square, and determining whether the program displays a selected number from a given list within the selected square. If the program displays the number pad after clicking on a given square, and displays a number once selected from the list without crashing the program, then the test is a success.

  Test 1: 4/22/22, Caleb Kimberlin, Passed during 2 games on Easy difficulty

  Test 2: 5/5/22, Amal Presingu, Passed multiple games on Medium difficulty

  Test 3: 5/5/22, Amal Presingu, Passed multiple games on Hard difficulty

- **2.6 - Play Again Button:** This unit is to denote the functionality of the "Play Again" button, which will be tested by a user/tester clicking on the "Play Again" button upon finishing a game of Latin Squares, and determining whether the program creates a new game of the same selected difficulty as the previous game. If the program creates a different game of the same difficulty as the previous game without crashing the program, then the test is a success.

Test 1: 4/22/22, Caleb Kimberlin, Passed after 2 games on Easy and Medium difficulty

Test 2: 5/5/22, Amal Presingu, Failed on all game difficulties (removed this button, as it overlaps with "return to menu" and doesn't serve any other purpose)

- **2.7 - Return to Menu Button:** This unit is to denote the functionality of the "Return to Menu" button, which will be tested by a user/tester clicking on the "Return to Menu" button upon finishing a game of Latin Squares, and determining whether the program displays the starting screen of the overall program. If the program brings the user back to the initial screen of the program after clicking on the "Return to Menu" button without crashing the program, then the test is a success.

  Test 1: 4/22/22, Caleb Kimberlin, Passed after playing 2 separate games on Easy difficulty

- **2.8 - Main Menu Music:** After the user fullscreens the game, music should start playing. It should then loop until the user selects a game difficulty. If the music plays, the test is a success.

  Test 1: 5/5/22, Amal Presingu, Passed through multiple runs

- **2.8 - Winning Music:** After the user successfully completes the board, celebratory music should start playing. It should then loop until the user returns to the menu. If the music plays, the test is a success.

  Test 1: 5/5/22, Amal Presingu, Passed on Easy difficulty

Signatures of the Testing Team

Caleb Kimberlin: Caleb Michael Kimberlin

Amal Presingu: Amal Presingu