INFT2012 - Lurgit Journal

May 2, 2018

Time Taken: 6 hours
Person responsible: Jacobus / Francois

To start off, we decided to create a simple C# program for rolling die so that we could play Lurgit with the goal of better understanding the programs requirements.
On the form we created three 100 x 100 picture boxes that will be used to simulate the die faces. We also added a button for rolling all three dice.
We then created a method to draw the required number of dots. This method has two integer variables as parameters. The first specifies the number to draw and the second specifies which picture box/die face it should be drawn on.

The next step was to create a function method to "roll" a die. This function method uses rand.next to generate and return a random integer between 1 and 6.

The player must have a way of holding/keeping some dice and rolling the others. To achieve this, we added two buttons at the bottom of every die picture box. One "Keeps" the die and the other "Rolls" the die. We created a bool variable for every die. These are ether true when the die is kept or false when the die will be rolled. The keep and roll buttons simply set this bool variable to either true or false. When the Roll Dice button is pressed there is a series of if statements that check if a die should be rolled or not that turn depending on the value of the bool variable.

The program was now capable of rolling and keeping die. We then played a few games of Lurgit to get a better understanding of the game and the requirements for the program. When playing the game, we generally had scores ranging from 10 –30 (once or twice it was high 30s). The highest score theoretically possible in Lurgit is if the player rolls a sequence bonus in the first two turns and a Lurgit bonus on the last turn every round. This will give the player a maximum score of 303 points. The probability of rolling a Lurgit bonus in a turn is 1/216 (1/6 * 1/6 * 1/6 = 1/216). Rolling a Lurgit in a round is very unlikely. In the games we have played so far, we have not rolled a single Lurgit bonus.

The problems we encountered:

- While testing the drawDie method we encountered an issue where the dots where not drawn in the expected location in the picture box. It seemed as if even though the picture box was specified as 100 x 100 in size, it was actually around 75 x 80.
- When testing the rollDie function method we discovered that it always returned the same number for all three dice.

These problems were solved by:

- It was only once we recreated the form and the picture boxes from scratch on a different PC that the drawDie picture box problem went away. We still did not know what caused the problem, but the dots where now being drawn in the expected location.

- To solve the rollDie function method problem, we added a small delay between die rolls (only fractions of a second) with System.Threading.Thread.Sleep().

Time Taken: 2 hours
Person responsible: Jacobus / Francois

Now that we have a better understanding of the game, we can design the form required for the program. We decided that a single form will be sufficient for playing a game of Lurgit. The form will show the scores for the players on the right. There will be three picture boxes for the die faces with “Keep” and “Roll” buttons underneath them. Above the picture boxes will be labels reading either rolling or keeping (so that the player can see which die will be rolled each turn). On the bottom of the form we will have a “Roll dice” and a “End Round” button. The form will also have a section at the top that keeps track of the players, turns and rounds.

We now altered the project form layout to suit our UI design. We then added code to the end round button method to tally the scores for that round and add to the players total game score. We still need to create methods to detect a Lurgit Bonus or Sequence Bonus. We also added text boxes to the form to keep track of the players previous round score and their total score. This could be improved later. We also made improvements to the roll die code by adding in loops instead of writing virtually the same thing three times in a row.

May 3, 2018

Time taken: 5 hours
Person responsible: Jacobus

I now had to add scoring for the end of the round (at this stage exluding Lugit and Sequence bonuses) to the program. I decided that the best place to tally the scores for the round would be when the player clicks the “End Round” button. First, I created an integer variable called iScoringNumber that would be used to check every die if it is showing the scoring number. I then created a for loop to loop for all three dice. The values rolled on every die is stored in an integer array called iDieRoll. In the for loop I added an if statement that would only be called if the iDieRoll value was the same as iScoringNumber. In the if statement, I then added the iScoringNumber to the iRoundScore. I then added the iRoundScore to the player total game score (iGameScore) after looping through all the dice. The game score is kept separate from the round score for testing purposes. Later on, we might only show the game score.

May 4, 2018

Time taken: 5 hours
Person responsible: Francois

I altered the code for the game to allow a second player to join in. We achieved this by creating a player class, holding the players required variables. I created a playRound function method so both players could have their turn before the round ends. The play round function was made using the existing code in the "End Round" button, while also

adding new code and altering some of the existing code. The playRound function returns a boolean that indicates whether the game has ended.

I created a function that would detect whether a Lurgit bonus is to be applied to the players score. I did this by creating a Boolean function method that would take the round number and the value rolled by each die as arguments. The function would then return true only when all the die values match the round number. The function would then be called in the "Roll" buttons code, only if all die were rolled. If the function returned true then 20 points will be added to the players game score as well as the sum of the die values, the player will no longer be able to roll the die and will have to end their turn.

The problems I encountered at this stage were:

- There was nothing set up to store the score and names of both players
- If both players were to play the game, there would be a lot of redundant code in the "End Round" button
- Needed a variable that stores whether the game has ended
- Needed a way to detect whether a Lurgit has been rolled and apply the bonus accordingly

These problems were solved by:

- Creating a player class to store information specific to each player
- Creating a "playRound" function method that would then be called multiple times
- Creating a Boolean variable that would store this information, which will be returned by the "playRound" function
- Creating a function method that would return a Boolean determining whether a Lurgit bonus should be applied

May 5, 2018

Time taken: 4 hours
Person responsible: Jacobus

We decided that like the Player class, we should create a Die class to hold the bDieKept (used to indicate if a die is rolled by the player or not) and the iDieRoll variables. I created the player class and then altered the program code to accommodate this change.

The problem I encountered at this stage was:

- I could no longer use simple for loops to loop through the die. The values for iDieRoll could no longer be in an array. I initially thought I could simply create an array of type Die to hold the values but found this too troublesome to implement.

This problem was solved by:

- I did some research online to see if I could find a way to create an array of an object but couldn't find a simple enough way to do this in the program. I then remembered from the lectures that it is possible to create an ArrayList for objects. I then created an ArrayList for the die called dieList and added the die objects to the list with dieList.Add(). I could now easily loop through the die objects using a foreach loop.

May 7, 2018

Time taken: 1 - 2 hours
Person responsible: Jacobus

To detect if the player has rolled a sequence bonus I created a function method called detectSequence that would take three integers (the die numbers rolled at the end of a turn) and return a boolean value of true or false. In the function method, I created an array called iDieArray to hold the die numbers rolled. I then sorted this array using Array.Sort(). I then used the middle number in the array (iDieArray[1]) to check if the value before it was one less and if the value before it was one greater. The function method then returns true if iDieArray[1] == iDieArray[0] + 1 && iDieArray[1] == iDieArray[2] – 1 is true.

The problem I encountered at this stage was:

- When testing the detectSequence function method I discovered that the player could simply keep all the die and click the “End Turn” button and the score for a sequence bonus would be awarded. This should not be allowed to happen.

This problem was solved by:

- Where the function method is called in the Roll Die button, I added an if statement to check if at least one die was being rolled before awarding a sequence bonus. Only if “!dDie1.bDieKept || !dDie2.bDieKept || !dDie3.bDieKept” is true does the program check for a sequence bonus.

May 8, 2018

Time taken: 3-4 hours
Person responsible: Francois

I had to make a few additional changes to make it possible to play the game with two players. I added 3 new buttons to the form for choosing the game mode. I then had to add an input box to the program for collecting the player names, to do this I used the code for creating input boxes from the lecture 6 demo. I added another two buttons that would end the turn for each respective player and moved the for completing a round to these buttons from the “End Round” button. The original “End Round” button became redundant and so I removed it. I added additional code that disables buttons depending on whether they are needed in a specific stage and/or game mode. This will prevent any bugs from accidental clicks. By disabling a player’s “End Round” button when it’s not their turn the program effectively passes control from one player to another.

The problems I encountered at this stage were:

- Clicking the “End Round” button would play the round for both players
- There was no way to differentiate what game mode it was
- Buttons being enabled throughout the game would cause bugs when clicked accidentally

These problems were solved by:

- Adding two new buttons to end round for each player
- Adding three new game mode buttons for each way you can play the game
- Changing certain buttons to default as disabled and to only be enabled when they are needed in the game

May 15, 2018

Time Taken: 1 hour
Person Responsible: Francois

I added a text box called txbxDisplayMessage, to display information to the user based on what is taking place in the game. The text box displays which players' turn it is, whether a bonus has been rolled and will display the player/s scores at the end of a game. It will also be used for other information as the program continues to develop.

May 16, 2018

Time Taken: 3 hours
Person Responsible: Francois

After watching the lecture on classes and objects I recreated the player and die classes and moved all the related variables and methods to the respective class. I assessed the rest of the code we have created up to this point. I then cleaned up unused methods, combined methods and created new methods for code that was used repetitively. I put all the code into regions to make the it easier to navigate, added comments to uncommented code and updated other comments.

When moving the methods for the Die to the Die class I encountered errors where the code was no longer recognizing the graphics objects. After extensive troubleshooting and help from Jacobus we managed to alter the methods so that they could be stored in the class.

Time Taken: 4 hours
Person Responsible: Jacobus

I now created a new method that will create a die rolling animation. This method will be called for each die in the die roll button click method. The method has a single integer parameter called iDieNo. This parameter is used to specify which die the animation is for. I first created a new Random object called rAnimation and an integer variable called iRandom to store a random number between 7 and 12. I also created another integer called iWait to store the amount of time to wait (in milliseconds) before redrawing the die face.

I then set up a for loop for the amount of time stored in iRandom. Inside the for loop I called the drwDie() method to draw the die face and wait for the amount of time specified in iWait.

Inside the loop I also incremented iWait by 20 to make the animation slow down over time.

The problems I encountered at this stage where:

- The dice are rolled in sequence (one by one). This does not simulate what would occur if a player physically rolled the die by hand.
- I realized the message box was not clearing after every roll
- There is no message indicating that the player has no turns left

These problems were solved by:

- To fix this problem I used Parallel.Invoke to execute the rollDie method for each die simultaneously. I learned how to use Invoke from docs.microsoft.com.
- https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/how-to-use-parallel-invoke-to-execute-parallel-operations
  For this solution to work properly I will need to add the drwDie, rollDie and rolling animation methods to the Die class. At this stage, the game does not work with Invoke since two or more of the operations are trying to use the drwDie method at the same time.
- I added txbxDisplayMessage.Clear() to the die roll button
- Added a message to txbxDisplayMessage after 3 rolls indicating that the player is out of turns

May 17, 2018

Time Taken: 3 hours
Person Responsible: Jacobus

While trying to use Parallel.Invoke to roll dice simultaneously I realised that placing the die roll and draw die methods in the Die class will solve the problem of multiple operations accessing the same method. The program will now access each die's own methods in the Die class, so the parallel operations will no longer be accessing the same methods.

Once I moved the draw, roll and animation methods to the Die class, I created a new class called RollDie. This method will be called in the PlayLurgit form (dDie1.rollDie). The RollDie method then calls the animation, roll and draw methods. The rollDie method has 4 input parameters. The first is type Graphics and will be used to specify the picture box the die face should be drawn on. The last three are type Random and are used to generate random numbers for the animation and roll methods.

May 19, 2018

Time Taken: 2 hours
Person Responsible: Francois

The goal today was to add the "play against the computer" functionality to the game. The first step taken to achieve this was creating a Boolean variable that would store that the game was in the play the computer game mode. I then copied the code used in the "two player button click" event and used in the play against computer button. I changed a few things in this code e.g. changing the player 2 name to be "Computer" instead of asking for a

name. The next step taken was to create a CompPlayer class that would store all the methods used to essentially make the computer play the game.

I created a method that would make the computer roll the dice three times and end the round. While testing this the computer performed the actions way to quickly for you to see what it was doing. To slow the actions down I added wait times before each step using the System.Threading.Thread.Sleep() method. The computer was now playing the game at a very basic level, the next step was to add strategy or decision making before the second and third roll for the computer.

After playing the game several times earlier we discovered that the optimal strategy was to roll for a sequence bonus in the first four rounds followed by rolling for score e.g. If its round five keep any fives rolled. I added an if statement before the second and third roll that would call a method called rollForSequence() if the round score was less than or equal to 4. I added an else if that would call the rollForScore() method if the round number was greater than 4. I then created the rollForScore() method which would essentially check if a die was not kept and matched the round number the computer would click the associated "btnKeepDie" button.

Time Taken: 4 hours
Person Responsible: Jacobus

After creating the CompPlayer class and having the Program simply roll all three dice on every turn, it is now time to add some strategy. While playing the game, Francois and I have noticed that it is better to roll for sequence bonuses every turn from round 1 – 4. Simply collecting the scoring dice in the early rounds doesn't amount to many points. For round 1, collecting 1's only awards the player with 3 points. In rounds 5 and 6, it is smarter to try and collect the scoring number. The reason for this is that if 3 fives are rolled in round 5 the player is awarded 15 points. That is significantly more than a sequence bonus. The player also still has the possibility of "accidentally" rolling sequence or lurgit bonuses.

To roll for sequences I first created a method in the CompPlayer class called rollForSequence. To keep track of the number rolled on the die and the which die it was rolled on I decided to create two parallel arrays. The first, called iDieRollArray, stores the numbers rolled on the dice. The second, called iDieNumArray, stores the die number. I can now sort both arrays at the same time by iDieRollArray using Array.Sort(iDieRollArray, iDieNumArray). This gives me a way of keeping track of what number was rolled on what die.

Now that the arrays are sorted I can check the die for two or three in a row. This is similar to how I designed the detectSequence function method. I then selected, using a series of if statements, which dice to keep, and which die to roll.

I also added a check for if the program rolls a sequence, to select the dice to keep that are not a 1 or a 6. This ensures that the program has the highest chance of rolling another sequence bonus.

The problems I encountered at this stage were:

- If some dice were already kept from a previous turn, the rollForSequence method would click the “Keep” button. This sets them to roll since they were already kept in the previous turn.

These problems were solved by:

- Adding a check to the if statements to only click the keep button if it is not already kept. I added a similar check for if the die that is already marked for rolling to ensure that the “Keep/Roll” button is not clicked if the die should be rolled.

May 20, 2018

Time Taken: 30 minutes
Person Responsible: Jacobus

To help the player see what the program is doing in play vs computer, I added a progress bar to the form. I also added a label above the bar to describe what step the program is up to for the player. Inside the playRound function, I increased the progress bar after every program action/ turn and updated the associated label to display information on where the program is up to. The progress bar is called pRoundProgress. To update the progress bar, I used pRoundProgress.Value = foo.

Time Taken: 2 hours
Person Responsible: Francois

I cleaned up some code in the CompPlayer class and ran a few tests of the “Play Computer” game mode. While running the tests the computer rolled a Lurgit bonus and clicked the disabled roll button three times before finally ending the round. To fix this I turned the two strategy methods for the computer into Boolean functions. The roll for sequence strategy would return true if a lurgit bonus was rolled. The roll for score strategy method would return true if all the die values matched the round score. When these functions were called in the playRound() method their return value would be stored in the bEndRound variable. This variable is used in selection statements to decide whether the computer will prematurely end its turn.

May 21, 2018

Time Taken: 2 hours
Person Responsible: Jacobus

I added a new button to the form called end session. The purpose of this button is to end the session for the player/s and show all the session statistics. This will include who won the session (calculated on total session score), total session score, games played, and games won for each player.

To achieve this, I added a series of if/else if statements that displays the statistics in a message box. The message box will be different depending on the number of players and who won the session.

The end session button also clears the session statistics for both players. This makes the reset button redundant, so I removed the Reset button from the form.
I spent some time going through all the program code to ensure there was no redundant code and that all code was commented. I also created regions to group code to make it easier to read.

The problems I encountered at this stage were:

- The stats for the second player is not shown if at some point the game changes to single player in the same session.

These problems were solved by:

- I changed the if statement for the end session button to only show the single player message if there was never a second player.
  I did this using the following statement:
  bIsSinglePlayer && pPlayer2.iGamesPlayed == 0

Time Taken 2 hours
Person Responsible: Francois

I added a "Game Rules" button that when clicked, would display a message box with the rules of the game. The first two paragraphs explaining the game I copied from the assignment description.

I changed the program so that it would display the round scores and the bonus scores separately. The combined game score is then only displayed at the end of the game in the "txbxDisplayMessage" text box.

I also edited the position of some of the controls on the form and renamed a few variables.

Lessons we have learned

The lessons we have learned while creating this program are:

- Spending more time in on the design phase of the project will make programming and implementing the design much simpler.
- Testing the program often and after every change made fixing bugs easier.
- Using classes for players and dice makes it easier to reuse and organise code.
- Having someone else test the program reveals bugs and other flaws in the game that we may not have noticed testing it ourselves. It also helps to identify areas where the UI design could be improved.