Department of Computer Science & CINSAM
NKU Summer Programming
Workshop 2015

Group Project

Work in groups of *up to 3*. Work alone ONLY if you have been successful in implementing most of the daily projects. Select one of the following. If you have alternatives in mind, talk it over with one of Gary or Richard. We must approve of any project you try as your project may not be realistic. If you have questions, ask!

1) Space Shooting Game
   Use the same basic idea as Pong and the Missile Launcher by creating a game where there are two players, one on each side of the screen, moving their spaceship up and down and firing at the other player's spaceship. If you like, you can allow the ships to move left and right as well as long as it doesn't move too far inward (no more than about 1/3 of the way from the edge). You might give each ship shields that, as the ship gets hit, the shields deplete, and a limited number of missiles. You can also add a gravity source somewhere near the middle of the screen that causes missiles to move off course. Alternatives are games like Space Invaders or Galaga.

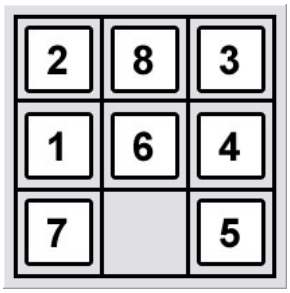
2) Breakout
   
   Have a bouncing ball and a paddle. Create a wall of bricks where a brick disappears if the ball hits it. Use a 2-D array for the bricks where an array element is true if that brick exists (hasn't been hit yet). This tells you whether to have a ball "bounce" at that position (it hits a brick) and whether to fill a rectangle or not (fill in for the brick, don't if there is no brick). The ball will bounce off of a brick, the top or the sides, but like Pong, if the ball gets passed the user's paddle, the ball is lost. Give the user some number of balls for the game (usually 3). Have bricks be worth different points and different colors. Use a Brick class to represent a brick. If the user clears all Bricks, you can have the user start a new level. Levels can be worth different points or have a reduced sized paddle or a faster ball to make it more challenging. Levels can also have different arrangements of Bricks by using a different initial array for that level.

3) Adventure Game
   Take the Fighter (from day 11) and make it into a true game. Use user input to control what happens in the game. The Fighter has an inventory of weapons, spells, positions and armor. Before a fight, the user can select what to use. After the fight, the user can claim any of the opponent's possessions for his or her own (some may be booby trapped!) Limit the amount the Fighter can carry so that the Fighter cannot just keep collecting items but must discard some. Before a fight, the user can rest, take a potion, cast a spell, etc. The user can also try to bribe the other Fighter to avoid fighting by offering one or more of his/her items (this may work or it may enrage the other Fighter). Invent your own rules!

4) Wheel of Fortune
   Redo hangman to be wheel of fortunate. Have 2 or 3 players. Before guessing a letter, the player "spins the wheel" (use a random number generator) which will be the amount of money the player earns if guessing a correct letter. If the letter appears multiple times, the user gets multiple amonts (for instance, if the guess is 't' and there are 3 't's, the user gets 3 times the amount on the wheel). The user can guess the answer at any time by typing in the full phrase (including spaces). If the user's input is a single character, then assume the user is guessing a letter, otherwise assume the user is guessing the full answer. The first user to guess the full answer wins the amount they have accumulated. Place the game in a loop and have the players play for some number of puzzles. Make the game text-based initially. If possible, try to convert it to one where the phrase is shown graphically. Note that guessing vowels do not win any money.

5) 3-D 4x4 Tic Tac Toe
Using the 2-D 3x3 Tic Tac Toe, expand it to be 3 dimensional and 4x4 so that there are 4 levels, each with a 4x4 grid. Although the program will be very similar to the earlier Tic Tac Toe program, you will find that there are a lot more winning combinations to check. Make it a human versus human game. Once running, if you want to try to make it human versus computer, give that a try. Once you have it running in a text-based manner, implement it using a JPanel and a paintComponent method so that output is done graphically.

6) 8 Puzzle
The 8 puzzle is a toy with 8 tiles and a gap so you can slide the tiles around to get them in order. Implement the 8 puzzle using a 1-D array of int values where the entry 0 is the blank. The user will input what tile they want to slide. Data verify that the selected tile is legal (that is, 1-8 and adjacent to the blank – for instance, in the figure to the right, the user could slide the 5, 6 or 7 only for this move). Implement this as a text-based game at first, output using System.out.println. Once running, try to change it to a GUI where the user can click on a tile with their mouse (you can use JButtons or you can use a MouseListener).



7) Who Wants to Be a Millionaire
Change the Millionaire program from textbased to a GUI with JButtons. Have 1 JButton for each of the 4 multiple choice answers plus a JButton for a lifeline and one to quit (also one to restart if you like). Use 3 different levels of questions, easy, medium and hard.

8) Deal or No Deal
In this game, 26 cases get random amounts and the user selects one to keep. Then, the user selects cases to open and based on the dollar amounts not yet seen (still hidden in cases), a banker offers the user some money. The user can either sell their case or keep playing. You will need 2 arrays for the 26 cases (an int array were cases[i] stores the dollar amount of that case and a boolean array were opened[i] is true if that cases has been opened and therefore no longer in use) and an int array of 26 elements storing the remaining dollar amounts so that the user knows what dollar amounts are still in play and so the banker can compute an offer. Make the game textbased initially and if possible, convert it to a graphical game or a GUI game where cases are JButtons to be clicked on.

9) Tron Motorcycles
This two player game has two players use the keyboard to change the direction of their motorcycle. The motorcycle leaves behind a wall. If either player runs into a wall, the player crashes and the other player gets a point. Play until one player has earned 10 points. To record the "wall", use a 2-D array. A 0 would mean "no wall", a 1 would mean a "wall created by player 1" and a 2 would mean a "wall created by player 2". Draw the walls in different colors.

10) Super Mario-like game
Create a game where you move a character around the screen using keys but can also "jump" the character over obstacles. Obstacles will be stored in an array to determine when you have reached one. Also, have the character jump to touch things to collect like coins.

11) Animated star field
Create a random series of stars by using the Star class and array of Stars. Animate the Stars by having them move off of the screen by moving each Star toward its nearest border (up, upper left, left, down to the left, down, down to the right, right, upper right). As the Star moves, increase its size slightly so that it looks like it is getting "closer". As a Star moves off the screen, create a new Star. As Stars are placed at random locations, the new Star will appear somewhere else. If two Stars are too close together, have them "collide" which causes one to disappear and the other to increase in size at a much faster rate.