CSC-348

Midterm Exam

During the exam window, you may not speak to another student about this exam. This applies to students enrolled in CSC-348 and students who are not. You may not speak to another faculty member other than your instructor.

This exam is administered as a take-home exam in order to facilitate creativity as well as completeness. Not adhering to the rules established on this page will break the Furman University Academic Integrity policy (https://www.furman.edu/academics/academic-resources/academic-integrity/).

Exam Timing:

This exam has *not* been written to take 48 hours to complete. However, you may take as much time in the 48 hours between

Monday, March 8, 2021, 8:00am and Wednesday, March 10, 2021, 8:00am

to complete and submit the exam. ***You are not to miss class or disrupt your other responsibilities as a student to complete this exam.*** Please be reasonable with your time and plan accordingly.

Writing the Exam:

- When context is clear, all answers must be written using complete sentences and proper punctuation.
- All solutions to exam questions are to be typed and any hand-drawn pictures must be neat (or drawn using software).
- Each question should begin on a new page; your answer must appear below the statement of the question. Maintain original question numbering.
- Do not include questions which you did not answer.

Submission:

- Your exam must be submitted as a PDF document.
- Your exam must be submitted by 5:00pm on Moodle.
- Late submission is via email at the cost of 1% of the exam for each minute late.

Grading:

- Each of the three questions will be weighted equally.

There are 3 sections to this exam: Design, Programming, and Mathematics. You must answer three (3) questions from at least two (2) of the three (3) sections.

Design

1. Consider a 2D side-scrolling game such as *Super Mario Bros* or *Braid*. Our goal is to design a small scene from a level in which the player must pass through the same scene 3 times. However, once a player uses a particular technique to traverse from a starting point to an ending point, the player may not use that same technique again.

In particular, you must design three distinct ways to go from a starting point to an ending point in your scene. These methods are not allowed to interact. As in, there is no way to go from start to end using a combination of two methods.

Your design must have a theme. Any equipment that a player may use must be reasonable for that theme and the game world. For example, in Super Mario Bros. 3, Mario had a raccoon suit that allowed him to fly for a given amount of time. In your solution, you must describe the scene, its theme, and any equipment required. You must also design clear reasons why the same traversal technique cannot be used a second time. For example, if the character can fly, they would be forced to lose the ability of flight for the other two traversals.

2. Games that feature many levels often rely on a lock-and-key mechanism to control the player's progress through each level. In some cases, these lock-and-key mechanisms are depicted as actual locks and keys. For example, in the Legend of Zelda, Link may pick up normal dungeon keys facilitating dungeon traversal. However, in later dungeons Link must also pick up a 'boss' key in order to enter the room and fight the boss.

For this question, your task is to design at least four (4) ways of using a sword as a key to a 'lock'. That is, the same sword will be used to unlock four (4) distinct 'locks' in the game. Please recall that locks in your game do not have to be actual locks. As an example of a physical lock you may not use in your solution is "A character may unlock a lock by inserting the sword's blade into an orifice in the wall."

For your solution, (1) clearly describe each lock, (2) how the sword is used to unlock each lock, and (3) how the use of the sword as a key is unique to the other unlocking scenarios. If necessary, provide appropriate details about the game and its associated world.

3. This question is inspired by *Challenges for Game Designers* by Brathwaite and Schreiber.

Electricity makes for great puzzles in video games since passing current from one side to another can logically result in some action being performed. Your goal is to create a prototype of a puzzle in which electrical current is passed from one side of the screen to the other. You may use any interface you like, provided that it requires spatial reasoning on the part of the player. You may also assume that if the player encounters the electricity, they will die.

Here are some questions to consider in your design. How do you get electrical current from one side of the screen to the other? What does a "success" look like? What are the ways that a player may "fail" or does the player simply keep trying until she succeeds?

You must submit a set of rules indicating the beginning state, objective, actions that the player may take, a prototype version of the puzzle, and answers to the questions from the previous paragraph.

4. Design a two-player, competitive, 2D video game in which the main mechanic is *picking up* items. Mechanics that modify the primary mechanic are acceptable. For instance, you could have players pick something up when they land on it, shoot it with a special gun, or require that the player have a backpack that they first must acquire in order to pick up the objects. Similarly, the player may pick up items that prohibit the other player from picking up items (or restricting types of items a player may pick up); consider 2D Bomberman.

You must choose a theme. It is strongly advised that you consider the theme as bringing different possibilities into play.

You must submit clear rules and a mockup of one level of the game.

Programming

5. Consider the Unity tutorial Roll-a-ball ‘game’ we implemented as a lab. Write a C# script that will attach to the main camera that will procedurally generate a set of walls to add to the playfield and thus simulate a 2D maze. For simplicity, you may focus on generating copies of one ‘maze wall’ prefab. For the purposes of this question, the maze does not need to be solvable (a path from start to finish does not need to be verified). No assets other than the camera, directional light, and exterior walls should be listed in the Hierarchy when the game is executed. In the inspector, we should be able to set the number of walls, but they should be limited to a reasonable number (around 20).

Copy and paste the formatted, commented source code as well as relevant screenshots demonstrating the success of your implementation.

6. In the Shoot 'Em Up game (part II), the author had you populate an array of size 10, then place Prefab enemies in each of the particular positions (#5 on Page 634 of the text). The goal of that setup is to generate enemies according to a particular probability distribution. That is, enemy 0 will be generated 30% of the time, enemy 1, 2, and 3 will each be generated 20% of the time, and enemy 4 will be generated 10% of the time. That approach is quick, dirty, and definitely poor software design.

Design and implement class `DiscreteDistribution` that simulates the roll of a weighted, n-sided die by implementing the following methods.

- The overloaded constructor takes a set of weights (doubles from 0 to 1); throws an exception if the number of weights is fewer than 2.
- A private method that computes the cumulative weighted probabilities as demonstrated in the table below.

Face Value	**Weights (probabilities)**	**Cumulative Weighted Probabilities**
1	0.1	0.1
2	0.1	0.2
3	0.1	0.3
4	0.5	0.8
5	0.1	0.9
6	0.1	1

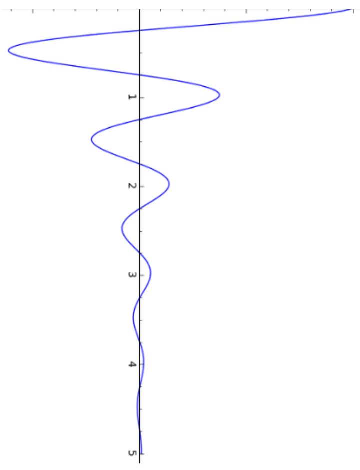
- The `generate` method rolls the die according to the given weights ***by referring to the cumulative weighted probabilities.*** Generate the corresponding roll of the die by generating a uniform random number and comparing to the cumulative weighted probabilities; e.g., values $[0, 0.1)$ results in a 1, $[0.3, 0.8)$ results in 4 using the cumulative weighted probabilities shown above.

The `DiscreteDistribution` class should compute and save the cumulative weighted probabilities; it should ***not*** save the weights.

Include this class in your implementation of the Shoot Em Up Plus game. That is, enemies should be generated according to the probabilities described in the book, but those pseudo-random values are generated by a `DiscreteDistribution` object.

Copy and paste the formatted, commented source code as well as relevant screenshots demonstrating the success of your implementation. The source code should include object creation and usage.

7. In the Shoot 'Em Up Plus implementation, create an enemy type with (a) velocity progressively slower, (b) strong easing out, and (c) moves according to a damped curve similar to the following.



The enemy will then proceed off the bottom of the board (and should be destroyed).

Copy and paste the formatted, commented source code as well as relevant screenshots demonstrating the success of your implementation.

If you answer this question, you may not answer question 12.

8. In the Shoot ‘Em Up Plus implementation, create a *Galaga* enemy type that moves similarly to an enemy in the 1981 game *Galaga*. That is, an enemy will fly in from the bottom left / right (or top left / right of the screen), circle around, and fall into ‘place’; please check out a video on youtube for a more revealing description.

For your implementation, you may not hardcode a fly-in path, but must use mathematics, a combination of interpolation functions, and / or easing. You may assume that only one Galaga enemy exists on the board at a time. Once the enemy falls into place, you are not required to move the enemy according to any protocol. For a bonus, you may shift the Galaga enemy from side to side a la classic Galaga.

Copy and paste the formatted, commented source code as well as relevant screenshots demonstrating the success of your implementation. The source code should include clear documentation of the mathematics of your fly-in path.

9. We can represent a dungeon from the Legend of Zelda (1986) as a sparse, directed graph with locks. Define pseudocode for an algorithm to identify the shortest path in a graph-based representation of a dungeon. You may assume an infinite number of bombs and complete knowledge of rooms, room contents, cracks in the wall (where bombs will be effective), etc. Please note that the resultant shortest path may contain cycles.

Provide evidence of the success of your algorithm on the Level I dungeon (from the text).

This is not an easy question and thus counts as two (2) questions on this exam.

Mathematics

10. Consider the horror game genre. Our goal is to define an easing function for a ghost that will provide progressive jump scares to a player. That is, the ghost should appear at a random location and progressively move toward (and/or away) from the player in a manner that would elicit a set of jump scares. You may assume that the ghost may travel through any physical object (walls, tables, etc.).

Your task is to define a mathematical function f for the easing, its corresponding graph in the interval [0, 1], and separate graphs of the velocity of f (derivative) and acceleration of f (second derivative) in the interval [0, 1]. Clearly indicate any points at which the first and second derivatives are undefined.

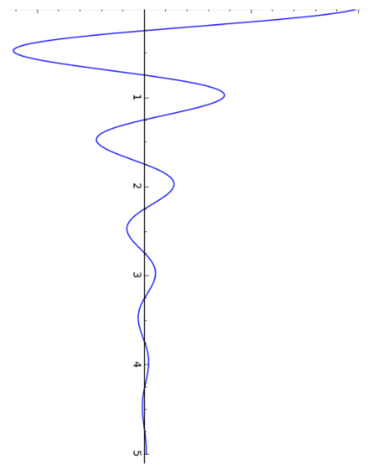
Clearly define the player and ghost as ‘start’ and ‘finish’ points; this must be made explicit. Last, describe (in words) the effect of the jump scare(s): when they occur, how they occur, etc.

11. We defined *linear* interpolation between two points, P_0 and P_1, using the formula

$$P_u = u \cdot P_1 + (1 - u) \cdot P_0.$$

The goal of this question is to define *circular* interpolation. That is, given points P_0 and P_1 that lie on a circle of radius r centered at (h, k), P_u would be a point on the circle between P_0 and P_1 along the minor arc for $0 \leq u \leq 1$. Derive the circular interpolation formula for direction angle $\theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$. Bonus for $\theta \in [-\pi, \pi]$.

12. Define a set of points that will result in a Bezier curve similar to the following graph.



Please provide the following in your solution.

- The coordinates of each point.
- In the Euclidean Plane, plot the points and label them such that an order of points is clearly implied.
- Draw an approximation of the Bezier curve as well as its convex hull.
- Provide a reasonable sequence of screenshots from the Interpolation lab demonstrating success of the points and the resulting Bezier curves.

If you answer this question, you may not answer question 7.