Computer Science 20/30 Project Plan	Name: Lucas Rauser

Project Idea:
I want to make a 3d rpg. Where you go through an open world where you talk to different npc's and make your way through a world collecting loot and items and fighting bosses that each have their own unique way of being beat.

Module	Software	Learning Resources (textbooks, websites)	Equipment	Timeline for completion.
OOP 2	Unity Visual studios	Unity textbook, internet, any old nevin videos using unity.	computer	
Recursive algorithms	Unity visual studios	Unity textbook, internet, any old nevin videos using unity.	computer	
Iterative algorithm 1	Unity Visual studios	Unity textbook, internet, any old nevin videos using unity.	computer	

Instructions:

Complete an overview of your project in the "Project Idea" section above. Review the CSE & ELT course curriculums(.pdf documents provided by your teacher) and find modules that fit with your project plan, list them on the left. You may need to adjust the "Project Idea" to meet the requirements of the specific outcomes outlined for each module(these will be used for evaluation of your project so it is crucial you meet all outcomes). List any learning resources(software, websites etc.). that you will use to complete your project across from their corresponding modules. List any equipment you need to finish

Computer Science 20/30 Module Final Evaluation		Name: Lucas	
Module: COURSE CSE3130: OBJECT-ORIENTED PROGRAMMING 2			
Outcome	**Demonstration/Self Assessment**	**Time**	**Evaluation**
1. explain and represent class and object interactions possible in OOP			
2. demonstrate OOP skills by writing algorithms employing an object-oriented approach to solving problems			
3. translate algorithms into source code, convert the source code into machine executable form, execute and debug, as required			
4. compare the results of the program with the intent of the algorithm and modify, as required			
5. demonstrate basic competencies			
6. create a transitional strategy to accommodate personal changes and build personal values			

COURSE CSE3130: OBJECT-ORIENTED PROGRAMMING 2

Level: Advanced

Prerequisite: CSE3120: Object-oriented Programming 1

Description: Students extend their knowledge of object-oriented programming (OOP). They add to their expertise in object-oriented design by using some of the techniques associated with the UML design approach and to their programming expertise by writing programs that explore association between classes. Students work with abstract classes, developing algorithms that employ the object diagram approach and programs that use templated classes, containment and inheritance to promote reusability.

Parameters: Access to appropriate computer equipment, software, the Internet and support materials. Access to OOP environment that encourages a formal treatment of objects.

Supporting Courses: CSE3010: Computer Science 3
CSE3110: Iterative Algorithm 1

Outcomes: The student will:

1. **explain and represent class and object interactions possible in OOP**
   - 1.1 outline the key properties of the OOP approach
   - 1.2 describe and demonstrate how coding can be reduced and responsibilities distributed through the appropriate use of polymorphism and inheritance
   - 1.3 describe and represent the relationship among the classes, objects, instances and methods including:
     - 1.3.1 inheritance
     - 1.3.2 association
     - 1.3.3 composition and aggregation
   - 1.4 describe and represent ways in which inheritance and polymorphism are promoted
   - 1.5 outline how static classes, polymorphism and inheritance may be used to hide and/or protect data
2. **demonstrate OOP skills by writing algorithms employing an object-oriented approach to solving problems**
   - 2.1 apply an object-oriented analysis and design model to decompose a data processing problem into a form that is accessible to an OOP approach by using:
     - 2.1.1 an informal domain analysis
     - 2.1.2 an informal use case analysis
     - 2.1.3 a general design model
   - 2.2 analyze a data processing problem and use a top-down design approach to transform a design model into a class diagram that represents the matrix of interacting classes required to solve the problem
   - 2.3 describe and represent the relationship among the classes; e.g., inheritance, association, aggregation, composition
   - 2.4 use an iterative and incremental approach to refine the architecture into appropriate class or object diagrams showing their relationships

2.5 analyze and refine the diagrams identifying the client/server relationship among the objects and determine the messages that need to be passed between objects and how the objects interface
2.6 draft an informal object message sequence indicating the flow of messages in the system
2.7 analyze and refine the object design by adding private methods, functions and data structures required to implement the various objects
2.8 test and modify, as required, the developing algorithm at each stage with appropriate data

3. **translate algorithms into source code, convert the source code into machine executable form, execute and debug, as required**
   3.1 demonstrate the use iterative and incremental approaches in the implementation, testing and maintenance phases of the software development process
   3.2 demonstrate the use of iterative prototyping or a similar approach to break the algorithm into appropriate sections for implementation
   3.3 use original (user-created) or pre-existing classes, as necessary, to instantiate the objects called for by the algorithm using an iterative and incremental approach
   3.4 as the classes are constructed, use the server classes to create the client classes establishing the client/server relationships called for by the algorithm
   3.5 test and modify the sections as required
   3.6 where appropriate, collaborate with other students to carry out OOP tasks
   3.7 create internal and external documentation
   3.8 analyze the program and eliminate syntax, run-time and logic errors by using appropriate test data for each programming task at each stage of development
4. **compare the results of the program with the intent of the algorithm and modify, as required**
5. **demonstrate basic competencies**
   5.1 demonstrate fundamental skills to:
      5.1.1 communicate
      5.1.2 manage information
      5.1.3 use numbers
      5.1.4 think and solve problems
   5.2 demonstrate personal management skills to:
      5.2.1 demonstrate positive attitudes and behaviours
      5.2.2 be responsible
      5.2.3 be adaptable
      5.2.4 learn continuously
      5.2.5 work safely
   5.3 demonstrate teamwork skills to:
      5.3.1 work with others
      5.3.2 participate in projects and tasks
6. **create a transitional strategy to accommodate personal changes and build personal values**
   6.1 identify short-term and long-term goals
   6.2 identify steps to achieve goals

Computer Science 20/30 Module Final Evaluation		Name:	
Module: COURSE CSE3110: ITERATIVE ALGORITHM 1			
Outcome	**Demonstration/Self Assessment**	**Time**	**Evaluation**
1. analyze and represent the nature, structure and utility of common iterative algorithms			
2. create and/or modify algorithms that use searches, sorts and merges to solve problems			
3. create and/or modify programs that use searches, sorts and merges to solve problems			
4. compare program operation and outcomes with the intent of the algorithm and modify, as required			

Outcome	Demonstration/Self Assessment	Time	Evaluation
5. demonstrate basic competencies			
6.			
7.			
8.			

COURSE CSE3110: ITERATIVE ALGORITHM 1

Level: Advanced

Prerequisite: CSE2120: Data Structures 1

Description: Students learn a number of standard iterative data processing algorithms useful for working with data structures such as arrays. These include an iterative version of the binary search, the three basic sorts—exchange (bubble), insertion and selection, and a simple merge. In the process, they learn when and where to apply these algorithms.

Parameters: Access to appropriate computer equipment, software, the Internet and support materials. Access to an object-oriented programming environment that encourages a formal treatment of objects.

Supporting Courses: CSE2130: Files & File Structures 1
CSE3010: Computer Science 3
CSE3120: Object-oriented Programming 1

Outcomes: The student will:

1. **analyze and represent the nature, structure and utility of common iterative algorithms**
   - 1.1 compare and contrast search, sort and merge algorithms
   - 1.2 explain the way in which search, sort and merge algorithms manipulate data
   - 1.3 describe the data structures required by search, sort and merge algorithms
   - 1.4 describe how search, sort and merge algorithms are implemented in a programming environment
   - 1.5 describe and represent iterative search algorithms including:
     - 1.5.1 linear search
     - 1.5.2 binary search
     - 1.5.3 compare and contrast how linear and binary searches manipulate data
     - 1.5.4 compare and contrast the data structures required and the computational efficiencies of linear and binary searches
   - 1.6 describe and represent basic iterative sort algorithms including:
     - 1.6.1 exchange sort; e.g., bubble sort, cocktail sort, gnome sort, comb sort
     - 1.6.2 selection sort; e.g., selection sort, strand sort
     - 1.6.3 insertion sort; e.g., insertion sort, library sort
     - 1.6.4 comparing and contrasting how different classes of sorts manipulate data
     - 1.6.5 comparing and contrasting the data structures required and the computational efficiencies of different classes of sorts
   - 1.7 describe and represent simple iterative merge algorithms
2. **create and/or modify algorithms that use searches, sorts and merges to solve problems**
   - 2.1 demonstrate the use of appropriate general design techniques for the programming environment being considered for implementation
   - 2.2 analyze and decompose the problem into appropriate subsections using the decomposition techniques appropriate for the chosen design approach
   - 2.3 evaluate subsections and identify any that may require some type of search, sort and/or merge algorithm, based on the nature of the data to be processed and the type of processing operations
   - 2.4 identify which algorithms are appropriate or required to search, sort and/or merge data

2.5 sequence the various subsections appropriately
2.6 test and modify the developing algorithm with appropriate data using a "fail-on-paper" process

3. **create and/or modify programs that use searches, sorts and merges to solve problems**
   3.1 convert algorithms calling for standard iterative structures into programs that reflect the algorithm's design
   3.2 use original (user-created) or pre-existing search, sort and/or merge algorithms appropriate to the data being manipulated
   3.3 utilize the appropriate operators, methods, functions or procedures required to carry out the standard algorithms
   3.4 use internal and external documentation
4. **compare program operation and outcomes with the intent of the algorithm and modify, as required**
   4.1 use appropriate error-trapping mechanisms built into the programming environment, as well as programmer-directed error-trapping techniques, to eliminate logic errors and debug the program
   4.2 compare the congruency between the outcomes of the debugged program and the original intent of the algorithm and modify both, as required
5. **demonstrate basic competencies**
   5.1 demonstrate fundamental skills to:
      5.1.1 communicate
      5.1.2 manage information
      5.1.3 use numbers
      5.1.4 think and solve problems
   5.2 demonstrate personal management skills to:
      5.2.1 demonstrate positive attitudes and behaviours
      5.2.2 be responsible
      5.2.3 be adaptable
      5.2.4 learn continuously
      5.2.5 work safely
   5.3 demonstrate teamwork skills to:
      5.3.1 work with others
      5.3.2 participate in projects and tasks
6. **create a transitional strategy to accommodate personal changes and build personal values**
   6.1 identify short-term and long-term goals
   6.2 identify steps to achieve goals

Computer Science 20/30 Module Final Evaluation			Name: Lucas
Module: COURSE CSE3310: RECURSIVE ALGORITHMS 1			
Outcome	**Demonstration/Self Assessment**	**Time**	**Evaluation**
1. analyze and represent the nature and utility of the recursive functions or procedures			
2. analyze and represent the nature, structure and utility of recursive search and sort algorithms			
3. create and/or modify recursive algorithms to solve problems			
4. create and/or modify programs that use recursion			
5. compare program operation and outcomes with the intent of the algorithm and modify, as required			
6. demonstrate basic competencies			
7. create a transitional strategy to accommodate personal changes and build personal values			

COURSE CSE3310: RECURSIVE ALGORITHMS 1

Level: Advanced

Prerequisites: CSE3110: Iterative Algorithm 1
CSE3120: Object-oriented Programming 1

Description: Students learn how to use a new program control flow mechanism called recursion. They then use this mechanism to write a number of basic recursive algorithms and programs such as a recursive version of the binary search, the quicksort and the merge sort.

Parameters: Access to appropriate computer equipment, software, the Internet and support materials. Access to an object-oriented programming environment that encourages a formal treatment of objects.

Supporting Courses: CSE3020: Computer Science 4
CSE3130: Object-oriented Programming 2

Outcomes: The student will:

1. **analyze and represent the nature and utility of the recursive functions or procedures**
   - 1.1 explain and represent the key features of recursive algorithms including:
     - 1.1.1 illustrate how recursive algorithms define themselves in terms of themselves
     - 1.1.2 illustrate the use and purpose of the base case in recursion
   - 1.2 describe and represent the "divide and conquer" approach to creating recursive algorithms
   - 1.3 describe and represent the interchangeability of recursive and iterative operations
   - 1.4 compare and contrast recursion and iteration highlighting:
     - 1.4.1 programmer efficiency
     - 1.4.2 space efficiency
     - 1.4.3 time efficiency
   - 1.5 outline the importance of recursion in creating dynamic data structures
   - 1.6 compare and contrast tail end and head end recursion
   - 1.7 explain and represent how the system stack (or equivalent structure) is used to carry out recursive operations
2. **analyze and represent the nature, structure and utility of recursive search and sort algorithms**
   - 2.1 describe at least four recursive algorithms used in dynamic data manipulation
   - 2.2 compare and contrast iterative and recursive approaches to binary searching by:
     - 2.2.1 describing and representing iterative and recursive binary search algorithms
     - 2.2.2 explaining the advantages and disadvantages of iterative and recursive approaches to binary searching
   - 2.3 compare and contrast at least two recursive sorts by:
     - 2.3.1 describing and representing the quicksort and the merge sort
     - 2.3.2 describing and representing the heapsort
     - 2.3.3 explaining the advantages and disadvantages of the quicksort, merge sort and heapsort

3. **create and/or modify recursive algorithms to solve problems**
   - 3.1 demonstrate the use of appropriate general design techniques to draft algorithms that use recursion
   - 3.2 analyze and decompose the problem into appropriate subsections using the decomposition techniques appropriate for the chosen design approach
   - 3.3 evaluate subsections and identify any that may require a recursive approach
   - 3.4 identify which recursive algorithms are appropriate
   - 3.5 sequence the various subsections appropriately
   - 3.6 test and modify the developing algorithm with appropriate data using a "fail-on-paper" process
4. **create and/or modify programs that use recursion**
   - 4.1 convert algorithms calling for recursive structures into programs that reflect the algorithm's design
   - 4.2 use original (user-created) or pre-existing recursive merge and/or sort algorithms appropriate to the data being manipulated
   - 4.3 utilize the appropriate operators, methods, functions or procedures required to carry out the recursive algorithms
   - 4.4 use internal and external documentation
5. **compare program operation and outcomes with the intent of the algorithm and modify, as required**
   - 5.1 use appropriate error-trapping mechanisms built into the programming environment, as well as programmer-directed error-trapping techniques, to eliminate logic errors and debug the program
   - 5.2 compare the congruency between the outcomes of the debugged program and the original intent of the algorithm and modify both, as required
6. **demonstrate basic competencies**
   - 6.1 demonstrate fundamental skills to:
     - 6.1.1 communicate
     - 6.1.2 manage information
     - 6.1.3 use numbers
     - 6.1.4 think and solve problems
   - 6.2 demonstrate personal management skills to:
     - 6.2.1 demonstrate positive attitudes and behaviours
     - 6.2.2 be responsible
     - 6.2.3 be adaptable
     - 6.2.4 learn continuously
     - 6.2.5 work safely
   - 6.3 demonstrate teamwork skills to:
     - 6.3.1 work with others
     - 6.3.2 participate in projects and tasks
7. **create a transitional strategy to accommodate personal changes and build personal values**
   - 7.1 identify short-term and long-term goals
   - 7.2 identify steps to achieve goals