# COMP 2404
# Take-home Final Exam

Due: Saturday, April 25 at 12:00 pm (noon)

Collaboration: Collaboration is **strictly disallowed**.
All submitted code must be original.

## The Tortoise Chronicles: Journey to Dragon's Hollow

*Note: You don't have to read the backstory to do the work. This part is just for fun.*

Trouble is brewing in King Aesop's peaceful realm. Handsome Prince Harold the Hare has come up with his most preposterous idea yet. He has decided that, in order to woo the fair Lady Gwendolyn and win her over, he will be braving the Mount of Dooom, on a quest to recover the Emerald of the Isles from the ferocious Dragon that guards it. He plans to present the Emerald to the fair Lady Gwendolyn in honour of her birthday. When Prince Harold asks the assembled throngs of citizens for volunteers to escort him to the Dragon's lair, he is naturally met with complete silence. He has thus decided to recruit über-geek Timmy Tortoise, Lady Gwendolyn's truelove, to accompany him on this hare-brained adventure. Timmy, not about to be outdone by a Hare, has foolishly agreed to go along. And to think that all Gwendolyn really wanted for her birthday was a replacement *Geek is Beautiful* T-shirt.

Of course, Lady Gwendolyn is well aware of the danger involved in this quest. The Emerald is believed to be hidden in a cave at the far end of Dragon's Hollow on top of the Mount of Dooom. The Dragon is known to pace in front of his cave and breathe fire upon and barbecue any intruders. Further, rumour has it that ever since Timmy Tortoise and Prince Harold attempted their race up the Mount of Dooom earlier this year, the Dragon has procured some hired muscle to help him protect his loot. Since the Orcs were already on contract to Middle-earth, the Dragon was forced to hire their smaller cousins the dwarf Orcs, also known as dorcs, along with their vaguely related and even more vertically challenged kin, the bear Orcs (borcs) and the pork Orcs (porcs). Aware that the two adventurers are hopelessly outmatched, Lady Gwendolyn must make sure that they survive their ordeal.

Sadly, Gwendolyn is not a ninja warrior type of heroine. But as an ace programmer in her own right, she is the brainy type. Before Timmy Tortoise's arrival in King Aesop's realm, Gwendolyn was the reigning five-time undefeated champion of the annual Programming Olympics. But one day, Timmy Tortoise came plodding into town to enter the competition and tied Gwendolyn for first place and the official title of Kingdom Geek. Not only was Lady Gwendolyn delighted to finally meet her equal, it was love at first byte. So to help her misguided friends, Gwendolyn has decided to program a model of Timmy and Harold's trek across Dragon's Hollow, where the cave sheltering the Emerald is located. In executing this randomized model multiple times to analyze all possible outcome scenarios of the quest, Gwendolyn believes that she can estimate exactly where and when to send the real ninja warriors that she has hired, so that they can slay the Dragon and rescue our heroes. Since Timmy Tortoise and Prince Harold have already left on their quest, packing plenty of carrots and lettuce for sustenance, time is of the essence for the development of this rescue plan. As a result, Lady Gwendolyn has asked you, Timmy's close friend and confidant, to help her with the development of the quest model.

# 1. Program Requirements

Your program will simulate a confined area called Dragon's Hollow, which is basically a flat, sunken area, surrounded by trees on three sides and a cave on the other. Timmy Tortoise and Harold the Hare will make their way from the left-hand side of Dragon's Hollow to the right-hand side, where the Emerald is located inside the cave. This cave is guarded by a Dragon and a bunch of dorcs, borcs, and porcs.

The goal of the program is for Timmy or Harold to make their way across Dragon's Hollow (moving from the left-hand side to the right-hand side), and rescue the Emerald, all without dying.

Your program will implement the following requirements:

1.1. The quest of our heroes (Timmy and Harold), and their battles against the hired fighters (dorcs, borcs, and porcs) guarding the Dragon's cave, take place in Dragon's Hollow. At start-up, your program displays the following:

1.1.1. Dragon's Hollow, represented as 25 characters wide (horizontal axis) and 5 characters high (vertical axis); the right-most edge of the Hollow represents a cave entrance in the middle three characters (see sample output)

1.1.2. Initially, the following quest players are shown:

(a) Timmy (represented by his avatar 'T') and Harold (with avatar 'H') are situated at the left-most edge of the Hollow, at different randomly generated vertical positions (you must use a random number generator function for this; one can be downloaded from here).

(b) The Dragon (represented by its avatar 'D') is situated at the right-most edge of the Hollow, at the midpoint on the vertical axis. The Emerald is not a quest player, but it can be represented as an asterisk '*', as shown in the sample output.

1.1.3. **Note:** The display must be stored as a class template specialization of a 2-dimensional array of `char`s. You **must** implement your own 2-dimensional array collection class template for this. In your implementation of this class template, the elements must be declared as a double pointer of type `T`, where the 2-dimensional array is represented as a dynamically allocated array of dynamically allocated arrays.

1.2. Timmy and Harold each begin with 15 health points. Timmy has a strength of 3 points and an armour of 3 points, and Harold has a strength of 5 points and an armour of 1 point.

1.3. **Note:** You will need a base class for all the different types of quest players. Your program must store a collection of the quest players that are located in Dragon's Hollow, and this collection must be a STL `list` of base class object pointers.

1.4. The program iterates until either (1) Timmy or Harold claims the Emerald, which happens when one of our heroes walks into the cave where the Emerald is located, or (2) both heroes are dead.

1.5. At each iteration, the following occurs:

1.5.1. With a probability of 60%, your program generates a new fighter:

(a) A new fighter has equal probability of being a dorc, a borc, or a porc. Dorcs are represented by the avatar 'd', borcs by 'b', and porcs by 'p'.

(b) Fighters are initially situated on the right-most edge of Dragon's Hollow, at a randomly generated vertical position in front of the cave.

(c) Fighters begin with 5 health points, and a randomly generated strength in the following ranges: dorcs begin with a strength between 6 and 8 points (inclusively), borcs with a strength between 8 and 11 points, and porcs between 4 and 5 points.

1.5.2. Your program loops over each quest player in the list. For each quest player, your program must invoke a function that **polymorphically** computes that player's next position. Quest players move according to these rules:

(a) Timmy and Harold move one horizontal position to the right, with a vertical position randomly generated between the same vertical position, one position up, or one position down.

(b) Fighters move one horizontal position to the left, with a vertical position randomly generated between the same vertical position, one position up, or one position down.

(c) The Dragon paces up and down in front of the cave entrance, moving up one position until it reaches the top position of the entrance, then reversing direction, until it reaches the bottom position of the entrance, then reversing direction, and so on.

(d) If the new position computed for a quest player is out of bounds of the Hollow, the player is positioned at an adjacent, valid position instead.

(e) If the new position is not occupied, the quest player moves to that position.

(f) If the new position is occupied by another quest player, a collision ensues and is handled as described below. Both quest players are still moved to that position, although only one avatar will be displayed.

(g) Any fighters that have reached the left-most edge disappear from the Hollow and from the quest.

1.5.3. The display is refreshed to show all the quest players, in their updated positions.

1.5.4. Handling collisions:

(a) If Timmy and Harold collide with each other, nothing happens.

(b) If a fighter collides with the Dragon or with another fighter, nothing happens.

(c) If a hero collides with the Dragon, the hero instantly dies.

(d) If a hero and a fighter collide, both quest players take a hit, as follows:

(i) If Timmy or Harold takes a hit, that hero's health is decreased by the damage inflicted by the fighter. The damage is equal to the fighter's strength, minus the hero's armour. When a hero's health reaches zero, he dies.

(ii) If a fighter takes a hit, that fighter's health is decreased by the damage inflicted by the hero, which is equal to the hero's strength. When a fighter's health reaches zero, it dies.

(e) If Timmy or Harold dies, his position is permanently marked with a '+' to indicate his grave.

(f) If a fighter dies, it is removed from the quest.

1.6. **Innovative feature:** In addition to all requirements stipulated above, you will implement a new, innovative feature as an add-on to the program. Your new feature must, at minimum, meet the following criteria:

1.6.1. It must require the implementation of a significant amount of additional code.

1.6.2. It must involve the implementation of several new classes that work together to implement the new feature.

1.6.3. It must be a *useful* feature in the context of the program, and not some filler or make-work feature.

1.6.4. To earn marks, the feature must be thoroughly described in your README file.

1.6.5. Examples of new features: the implementation of a medium-complexity design pattern (not one that requires only a couple of new classes), the creative use of a graphics library already in the default VM (for example, `ncurses`) to display the output, the addition of several new types of creatures, each with new functionality, etc.

**Note:** You may make very minor, cosmetic changes to the requirements in order to accommodate this new feature. For example, increasing the size of Dragon's Hollow is fine, as long as the outcomes remain the same. Specifically, there must be executions of the program where Timmy wins, executions where Harold wins, and executions where both heroes die.

1.7. Sample output: You will find portions of a sample execution here.

## 2. Constraints

Your program must comply with all the rules of correct software engineering that we have learned about during the semester, including but not restricted to:

2.1. The code must be written in C++98, and it must compile and execute in the default course VM. It must not require the installation of libraries or packages or any software not already provided in the default VM.

2.2. Your program must follow correct encapsulation principles, including the separation of control, view, entity, and collection object functionality.

2.3. Your program must not use any classes, containers, or algorithms from the C++ standard template library (STL), unless explicitly permitted in the instructions.

2.4. Your program must follow basic OO programming conventions, including the following:

2.4.1. Do not use any global variables or any global functions other than `main()`.

2.4.2. Do not use `struct`s. You must use classes instead.

2.4.3. Objects must always be passed by reference, never by value.

2.4.4. Existing functions must be reused everywhere possible.

2.4.5. All basic error checking must be performed.

2.4.6. All dynamically allocated memory must be explicitly deallocated.

2.5. All classes must be thoroughly documented in every class definition.

## 3. Submission

3.1. You will submit in *cuLearn*, before the due date and time, the following:

3.1.1. A UML class diagram (as a PDF file), drawn by you using a drawing package of your choice, that corresponds to the entire program design.

3.1.2. One `tar` or `zip` file that includes:

(a) all source and header files

(b) a Makefile

(c) a README file that includes:

(i) a preamble (program author, purpose, list of source and header files)

(ii) compilation and launching instructions

(iii) a comprehensive description of the innovative feature that was implemented, including what the feature does and what classes are involved in providing the functionality

3.2. **LATE SUBMISSIONS WILL NOT BE ACCEPTED, FOR ANY REASON.** This includes technical and/or connectivity issues. Do not wait until the last day to submit.

3.3. Extensions to the due date and time will **not** be granted, for any reason.

3.4. Only files uploaded into *cuLearn* will be graded. Submissions that contain incorrect, corrupt, or missing files will receive a grade of zero. Corrections to submissions will not be accepted after the due date and time, for any reason.

# 4. Grading

## 4.1 Minimal criteria

In order to be graded according the categories listed in section 4.3, your program must meet the following minimum criteria:

- The program must be implemented in C++98, and it must compile and execute in the default course VM.
- A correct Makefile must be provided.
- The program must successfully complete at least five (5) separate and different (randomized) executions that meet all requirements and constraints described in sections 1 and 2.

**Submissions that do not meet this minimum criteria will not be graded and will earn a grade of zero.**

## 4.2 Rubric-based grading

The final exam will be assessed in accordance with the grading categories listed in section 4.3. Each category will be graded based on a 6-point rubric. Some categories are worth double the weight of other categories, so their value is 12 points each. These will still be assessed according to the 6-point rubric, but the values will be doubled.

The 6-point rubric is defined as follows:

- [6 points] Excellent (100%)
- [5 points] Good (83%)
- [4 points] Satisfactory (67%)
- [3 points] Adequate (50%)
- [2 points] Inadequate (33%)
- [1 points] Poor (17%)
- [0 points] Missing (0%)

## 4.3 Grading categories

The program will earn an overall grade out of 60 marks. This will be the sum of the grades earned for each of the following categories, based on the 6-point rubric described in section 4.2:

4.3.1. **UML [6 marks]:** This criteria evaluates the UML class diagram corresponding to the design and implementation of your entire program. The diagram must be drawn by you, using a drawing package, and it must be submitted as a PDF file. Other formats will not be graded. Your UML class diagram must comply with the UML format and conventions covered in the course material.

4.3.2. **Object categories [12 marks]:** This criteria evaluates how effectively the functionality of your implementation is separated into control object(s), view object(s), entity objects, and collection object(s), and how it follows the correct software engineering rules of data abstraction and encapsulation.

4.3.3. **Code quality [12 marks]:** This criteria represents the overall code quality. This includes, but is not limited to, how your classes are implemented, how your functions are designed, how the code is written, how well the principle of least privilege is applied, the presence of constructors and destructors where appropriate, how the code complies with all constraints listed in section 2, etc.

4.3.4. **Polymorphism [12 marks]:** This criteria evaluates the effective development and usage of polymorphism in your code, specifically as it relates to the behaviour of the entity objects, as described in the program requirements in section 1.

4.3.5. **Innovation [6 marks]:** This criteria evaluates the innovative feature that you will add to your program. This feature must be substantial, in the sense that it must add several new classes to your program, and these new classes must work together to implement the new feature. The feature will be evaluated based on how much additional code is implemented, and the creativity and the programming sophistication shown in selecting and implementing this feature. Features that are not described in the README file will earn a grade of zero.

4.3.6. **Class template [6 marks]:** This criteria evaluates the effective development of a new class template representing a 2-dimensional array of `T` objects, and its usage to hold the 2-dimensional array of `char`s displayed on the screen, as described in the program requirements in section 1.

4.3.7. **Packaging [6 marks]:** This criteria evaluates the packaging of the code into an archive file. This includes, but is not limited to, the correct separation of the code into header and source files, the presence of complete and correctly formatted Makefile and README files, and the absence of duplicate, additional, and/or unneeded files.