CSC 100 Final Project

Over the final four weeks of this class (leading up to the Thanksgiving break), you will work on a project of your own design. Students will team up to produce some creative software product using either Snap! or Python, such as a game, educational program, tool, interactive animation, etc. Python programs can either be text-based or graphical (turtle graphics or sprites). *Groups can be either two or three students*, and three-student groups will be expected to do proportionally more work than two-student groups.

The goal of this assignment is to give you some experience with developing a reasonably complex application. Your experience with programming so far has been limited mainly to guided labs and homework assignments; whether you realize it or not, you've already gained a lot of programming knowledge over the course of the semester. This project is designed to let you use all of this new knowledge to produce something that is interesting, useful, and challenging for you. We want to unleash your creativity as much as possible, so the purpose of your project will really be up to you and the other people on your team. But keep in mind that this is a computer science class and not an art class: your first priority is to spend time designing algorithms and writing code, not just making cool graphics and sprites. See below for a list of basic project requirements.

At the last lab meeting of the semester there will be a "project showcase" in which students present and demonstrate their projects for the class. Students will vote on the projects, and the winning project will receive a "Student's Choice Award."

Timeline and Due Dates

The project will proceed in stages, with regular deliverables and activities. For all major deliverables there should be one submission per group, but make sure all project team member names are included on any written submission. The one exception is the teamwork reflection that is due from each student individually (see below under “Final Deliverables”). The following table summarizes due dates. Unless otherwise specified, all deliverables are due before class or lab (i.e., they are due at 10:00 AM).

Date(s)	***Activity/Deliverable***
Oct 11-20	Initial team organization
Fri, Oct 27	In-lab project proposal presentations and discussion
Wed, Nov 1	Written project proposal due
Fri, Nov 10	Progress report due (5:00 PM)
Fri, Nov 17	Final project demos and presentations
Wed, Dec 6	All project materials due (due before Final Exam at noon)

Details for each part are described below.

Graphics, Sound, Copyright, and Licenses

Most people will make a program with some graphics and/or sound component. While it is tempting to think you can just grab things off the Internet, you *may not use any media (graphics or sound) that you do not have the rights to* (in this class or anywhere else!). This is very important, and I will not accept any project that includes material that you don't have the legal right to use. If you submit something that includes such items, you will get a zero on the project. You are required to document the source for all of your project resources in your final report. Fortunately, there are a lot of generous people out there that create graphics and sound that you can use for free! Here are some examples:

- "Game Art 2d Freebies": http://www.gameart2d.com/freebies.html
- itch.io free game assets: https://itch.io/game-assets/free (includes sound as well as graphics)
- OpenGameArt.org: https://opengameart.org/

Make sure that anything you use is specifically labeled as free or carries a license like a Creative Commons license. Note that you must have the rights to both the specific artwork and the characters. Just because you draw your own picture of Spiderman doesn't mean that you have the right to use that!

Project Proposal: Presentation and Written Proposal

We will spend some time in class organizing students into groups with similar interests, and the first major thing that teams must do is a proposal presentation. Each team will have a total of 8 minutes, and should plan for 3-4 minutes to present their ideas and about the same amount of time for discussion/feedback. The purpose of these presentations is to help you focus your ideas, and to get good and creative ideas from other students. *All students are expected to be engaged in these discussions* – you will get a small participation grade during these presentations, so if you don't contribute to the discussion of some other group's project you will not get any of these points.

A written project proposal is due Wednesday following your presentation. This will give you the opportunity to incorporate any good ideas you get from the proposal discussions. This document should be one to two pages, double-spaced. Start with two or three paragraphs that describe the overarching purpose of your project. Is it a game? Educational software? A utility? A sound-based application? After describing the main purpose, discuss the "scope" of the project: the types of things users will and will not be able to do with it. This can include a basic plot line for a game or animation, a list of options for a utility, and so on. Finally, prioritize your project scope: list things that it must do, and things that you'd like for it to do. You need to submit *something that works*, so prioritizing this way will ensure that you can submit something even if you don't get to all of your (hopefully a little ambitious) plans.

This is a mostly non-technical document and should describe big ideas more than the technical details of how it will be done. I will discuss your project proposal with you if I have questions about feasibility: in particular, I'm interested in gauging the level of difficulty of the project, the estimated workload, design choices, and potential pitfalls. *Note:* although you will be able to tweak your project idea after your proposal has been submitted, the gist of it should stay the same. Make sure that you're happy with your idea since you'll be spending a good bit of time with it.

Progress Report

This document should again be one to two pages, double-spaced. You should describe what progress you have made towards completing your project – among other things, you should include:

- the general breakdown of work among the members of your project group,
- what milestones you have set and which have been met,

- what problems you have encountered and how you overcame them,
- whether (and how) your project has changed from when you first described it in the project proposal, and
- what is left to be done.

This should be a reasonably technical document. Feel free to talk about particular functions or blocks if it helps you communicate your solution. You can also briefly discuss your algorithms for particularly complex problems. Bear in mind that by this point, you should have completed approximately two thirds of your project.

Final Project Demo and Presentation

Your project should be finished and working before the project presentation. You should create a presentation and demo for your project. Each team will have approximately 7 minutes, and each team member must have a significant (at least 2 minutes) speaking role in this presentation. In addition to demonstrating how your program works, you should talk about individual responsibilities within the project, what challenges you encountered, and what you learned.

You should save your project code and presentation PowerPoint on a USB flash drive so that you can efficiently bring up your materials during your presentation. While you are not required to use the following format, it is strongly suggested (if you want to deviate from this, it would be a good idea to discuss your ideas with the instructor first):

- Title Slide – Project Title, Team Members, Date
- 1 Slide on project purpose and why you chose it
- 1-4 Slides with an overview of the implementation design
- 1 Slide summarizing what challenges you encountered, and how you dealt with them

And then demonstrate your project! Note that due to the number of projects that must be presented, your time is limited – if you create a long animation, you might have to create a special version for the presentation that extracts some interesting smaller pieces that can fit into the time limits.

Project Submission

The final deadline for the project is **Wednesday, December 6, 2017 at noon**. Note that this is the time for the final exam, so plan ahead! Each team will submit **three items:**

- Your program – if you use Snap!, you can use the "submit" option under "the G" menu. If you use Python, you should submit your Python files and any support files (graphics, etc.) in Canvas.
- The PowerPoint slides from your presentation.
- A report that includes one page with basic instructions for how a user would run and interact with your program, and 2-3 pages that give a summary of significant programming components of your project. This second part should describe your basic design: how you partitioned the work between program pieces (sprites, blocks, functions, …), what functions or blocks you defined and why they made sense as abstractions. How did you use lists in your project? Talk about how your project works on an abstract level, in terms of programming components working with each other. You don't need to go into detail, because I will have the code too. You must include a section in your report where you describe the source for any artwork or sound used in your

project. Did you create them yourself? If you downloaded freely-usable materials from the Internet, include a URL for the website that provided the material.

Only one person should submit these three items for the team, but make sure you have all team member names on the report. Some of your projects may be huge in terms of file size. Make sure that you use efficient, compressed formats for things like images and sounds whenever possible.

Finally, *each individual should submit a short report* (1/3 to 1/2 page) describing how you felt your team functioned. Did you think there was sufficient and effective team communication? Did everyone contribute a fair share of the work? What lessons did you learn about teamwork that will affect how you approach team projects in the future?

Basic Project Requirements

- There must be a clear way to start your program. In Python, it is recommended that you have a function named run(), so that typing run() in the Python Shell runs your program. In Snap!, using the green flag is the recommended approach.
- If you create a game, it can be either a creative game or a game with a purpose (a mind-expanding puzzle game, training game, educational game, etc.).
- All programs should be interactive. If you do an animation, it should involve significant user interaction – it's a program, not a movie!
- "Style" points will be awarded for things like well-designed code. Poor structure can include things like duplicated code, needlessly complicated organization or program flow, and so on.
- Create some of your own functions and blocks. Proper abstractions are a must!
- Use lists in your project in some way.
- You cannot work alone, but can either work in pairs or groups of three. Expectations will be higher for larger groups.

Project Ideas and Sources of Inspiration

Here are some ideas of both types of projects and specific ideas that should get you thinking about possibilities:

- Game-based AI Agent: Write a program that can play a game involving chance, like Yahtzee, Monopoly, or poker.
- Puzzle Solver: Write a program that can solve and/or generate sudokus, word searches, crossword puzzles, etc.
- Tutorials: Write a program that teaches the user about some topic (a topic from computer science, music, math, …)
- Art Generator: Write a program that can generate images, music, etc.
- Modeling Algorithms: Write a program to simulate and analyze different strategies for solving a problem, such as different ways to serve customers at a store (one line for multiple registers vs multiple lines?), ways to board a plane (front to back or window to aisle?), ways to pack a truck (as packages come in or order largest to smallest?), …

Do not use another person's project to start creating your own – your project should be entirely your own creation. Nevertheless, getting inspiration from other projects, programs, etc. is encouraged. The

Berkeley class that this one is patterned after has their project students make videos that demo their projects – here are a few that look particularly good:

- Texas Holdem (Snap!): https://www.youtube.com/watch?v=o_EcccPTqio
- Minesweeper (Snap!): https://www.youtube.com/watch?v=fsOn4eBn1Og
- Monopoly (Python, text based): https://www.youtube.com/watch?v=fHxgIizk3Wg
- Snake (Python, text based): https://www.youtube.com/watch?v=2ipHzFiaelQ

And if you want to see a really ambitious project and video, check this one out:

- Sudoku (Snap!): https://www.youtube.com/watch?v=_yAzgt4AGbY

A Word of Warning

Getting to design projects of your own can be exciting, and it is very easy to underestimate how long it will take to accomplish a particular goal. Remember: although we are happy to help you bring your idea to life, you won't have lab-like guidelines for making this happen. It may take a lot longer to make your project than you think! That being said, don't hold back if you think you can make something truly grand. We're here to help you if you've got an idea that you love.

Project Tips

- Save and checkpoint often. This may entail creating multiple versions of your project to act as backups.
- Begin with functionality, and then start adding assets (custom-made sprites, imported lists, sounds, music, etc.) only ***after*** you've gotten your scripts working correctly. For lists, define test lists in your program that you can use for testing. In Python these can just be lists assigned to a variable, and in Snap! you can use either the "list" reporter block, or create a script that constructs a test list.
- Test each of your functions/blocks thoroughly. You don't construct a skyscraper and then start checking the first floor for structural problems. Test each part as you go along, and build from a solid foundation.
- Choose *descriptive* variable, function/block, sprite, etc. names. Follow sensible naming conventions, and if you work independently with the intent of merging your work, make sure you plan out names of objects so that the merging goes smoothly.
- Minimize the number of "global variables" that you have (variables that can be seen and/or modified by all of the sprites). These tend to cause hard-to-track-down bugs. Use local variables or "script variables" whenever possible.
- If a complex function/block isn't working correctly, start pulling pieces out of it and test each one individually (aka "Unit Testing"). Give the isolated block a set of known inputs, determine what the output should be, and *then* compare the results.
- One more time: ***save and backup often***!