CS 132, Winter 2022
Final Programming Project: Custom Simulator (60 points)
Due in six milestones on 2/22, 3/4, 3/9, 3/11 and 3/12

For this assignment you will work in a group of 2-3 students to implement a simulation of your choice. Your program will be implemented in C++, using what we have learned this quarter. This is meant to be an opportunity to create something that you find interesting and exciting. **Your program must meet all of the requirements below**. However, you are welcome to add as much additional behavior as you like.

Concept:

Your program must allow a user to specify some options and then simulate a situation using those parameters. The simulation must run as an animation in a graphical user interface. You can allow the user to interact with the simulation while it is running or you can just have it run until finished and allow the user to restart it with new parameters

Whatever the topic of your simulation, the simulation area should be divided into cells with integer coordinates, similar to the Critter world from project 5.

Implementation Requirements:

1. *Your program must allow a user to specify some options and then simulate a situation using those options*.

   You can allow the user to interact with the simulation while it is running, or you can just have it run until finished and allow the user to restart it with new parameters.

2. *The simulation must run as an animation in a graphical user interface.*

   You may include GUI inputs to allow the user to specify parameters in the GUI, or just include constants that can be altered in one of your files.

3. *The simulation state should be stored in a class that maintains and alters a grid like structure (vector of vectors or array of arrays) of objects*.

4. *A separate class should create the graphical interface and animate it by redrawing its contents on the screen every set amount of time.*

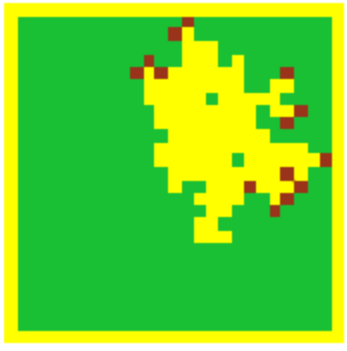
   This class should figure out what to draw by calling member functions on your grid model class.

5. *The objects stored in the grid must have an inheritance relationship.*

   Your **inheritance hierarchy** must have a **minimum of three levels and six classes**. The subclasses must be more specific instances of their superclass(es). Your grid must be able to contain instances of all of them.

Simulation Ideas:

Your program can simulate anything that your group likes as long as it is school appropriate. Here are some ideas (although **you do not have to use any of them**):

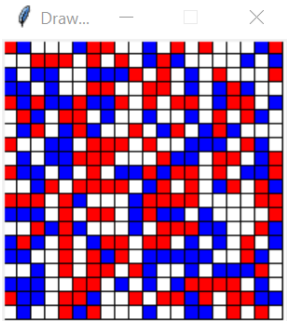


Fire:

Computers are commonly used to simulate natural disasters and events like fires and floods. This is super common in computer gaming to make games more realistic and interesting. However, it also has many applications in preventing and controlling real life disasters. For example, fire simulations are also used to predict how fast and to what locations fires will spread so that people in danger can be evacuated.

To simulate how fire spreads, use a grid of squares, each representing a square meter of ground. Some of these squares may contain trees, others vegetation or vegetation debris, some nothing, some fire, etc. Each of these will burn (or not burn) at a different rate. Fire can't spread if there is nothing to burn and it will eventually consume all the fuel in any location that gets lit on fire. However, if the area around the fire has burnable contents the fire may spread there. The speed, direction and probability of this will be determined by the humidity, wind and other factors. The spread can be altered by the addition of fire retardant and fire breaks.

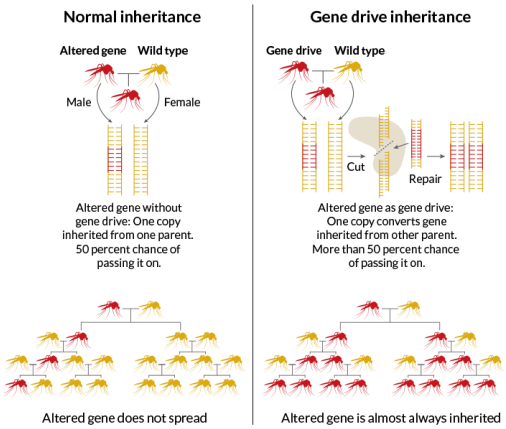
Segregation:



Racial segregation has always been a pernicious social problem in the United States. Although much effort has been extended to desegregate our schools, communities, and neighborhoods, the US continues to remain segregated by race and on economic lines. Why is segregation such a difficult problem to eradicate?

In 1971, the American economist Thomas Schelling created an agent-based model that might help explain why segregation is so difficult to combat. His model of segregation showed that even when individuals (or "agents") didn't mind being surrounded or living by agents of a different race, they would still choose to segregate themselves from other agents over time! Although the model is quite simple, it gives a fascinating look at how individuals might self-segregate, even when they have no explicit desire to do so.

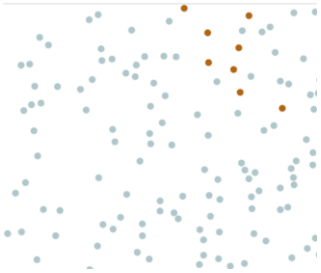
You can find more detailed information about this model here.



Crispr:

Simulate how a group of creatures with genes edited to stop their offspring of one sex from being able to reproduce would alter the population of that species and the other species in the ecosystem. This may sound odd, but it has been proposed as a potential plan in multiple places. New Zealand would like to get rid of an invasive rat population to save the native birds. Florida is trying to get rid of disease spreading mosquitos by releasing genetically modified mosquitos using a similar method. Simulate what would happen to the modified animal population and the rest of the animals of that species. Simulate how this decline in population would alter other species in the ecosystem. Other creatures will have access to more food if they were competing with the declining population animal and therefore their populations will increase. However, animals that depend on the declining species as a food source will get hungrier and may starve.

Contagion:



Simulate how Covid (or another disease or idea) spreads through a population. You must simulate people moving around in the world and every time a person meets another there is a certain percentage chance that, if they are sick, they will infect that person. Depending on how much (and how far) people move, how far they stay from others and at what percentage chance disease will transfer the rate of disease can go up or down. You can find examples of how different interventions affect transmission rates here.

Development Strategy:

Start by designing your inheritance hierarchy. What will your base class be? What functionality will all classes need? This is good functionality to add to your base class. Make sure you include enough information to allow you to alter the simulation state over time.

Next, implement the classes in your inheritance hierarchy. These will be somewhat similar to Critters from project 5.

Now create your model class that represents the simulation world (grid of objects). Add functionality to this to allow you to print out a text representation. Add functionality to this to allow you to update the state over time. This should be done primarily with calls to member functions of the objects stored in the grid.

Finally, implement your graphical user interface so that you can watch your simulation animate.

It will be very important to test your code **incrementally**. You will want to write several text main programs as you develop your classes to make sure that they are working correctly.

Extra Features:

Your group will receive **2 extra credit points** for each of the following additional features you implement:

1. Creating a Doxygen API for your classes from your comments

2. Adding interface elements (textboxes, buttons, checkboxes, radio buttons, etc) to allow the user to input simulation parameters in the GUI.
3. A save and load feature that allows the user to save the parameters of the simulation to a file and then on a future run load a file generated by a previous save to run the simulation with those parameters again.

Do you have other ideas that you think would make good extra features? Let Allison know and you may be able to use them for extra credit as well.

Style Guidelines:

This is a large assignment that encompasses almost everything that we have covered in class this quarter. Keep all previous style guidelines in mind as they will apply here as well.

In particular, make sure you follow proper **object-oriented** programming style. You should **encapsulate** the data inside your objects, and you should not declare unnecessary data members to store information that isn't vital to the state of the object. Take advantage of **inherited functionality** and implement **template**d code when necessary to eliminate redundancy.

As always, follow past guidelines about proper indentation, spacing, identifiers, and localizing variables. Make sure to use good Boolean Zen, and factor your conditionals.

Place comments at the beginning of each header file including your group information and documenting that header's behavior, on top of each member function header, and on any complex code. Remember that your main public comments should be in your header files not in your cpp files. However your cpp files should contain comments on the file header as well as on all member functions.

Make sure that you remove any debugging code before submission.

Group Work Guidelines:

You will work with one or two other students to complete this project. Group work can either be way better than working alone if your group works well together or way worse if it doesn't. In order to ensure that everyone has a good experience you will be required to complete and submit documentation of the following activities to receive full credit:

1. Spend **at least 6 hours** coding together. This means writing or debugging code while looking at the same screen. It includes discussing the best way to implement a function or feature and actually writing the code together. **The more time you spend coding together the more you will get out of this project.**
2. If your group is going to spend time coding on your own this should be done by all group members not just one. This means that there shouldn't be a situation where the group members code two classes together and then one group member writes four other classes on their own. If there are four other classes to write and you need to complete them separately these should be split evenly between the group members. Split tasks based on the amount of work not number of classes as some will be much harder than others.

   Although tasks should be split equally, it is just fine if group members end up collaborating more on one class versus another to help debug, alter the interface or otherwise perfect the code. The goal is just that everyone is contributing.
3. **Check in regularly** (at least every 3 days) with your other group members about your project progress.
4. Every group member must keep a log of the following:
   a. The time they spend working on the project.
   b. The classes/functions that they are the main author of.
   c. The additional classes/functions that they helped debug.

   You will have the option of using a GitHub repository to maintain your code for this project if you would like. However, this is entirely optional. If you choose to do this you may find it simplest to put the above information in your commit messages. That way, when you are ready to submit, you have it all in one convenient place.
5. **Let Allison know right away if your group is having trouble meeting these guidelines**.

Your group's project will receive a percentage grade as specified later in this document. By default, this **grade will be shared among all group members**. However, if particular group members contribute a large amount more or a large amount less than their partners, they may be subject to an individual multiplier between 0.5 and 2. Refusing to meet,

failing to communicate, not doing one's share of work, or other negligence may result in a multiplier less than 1.0. This will only be done in circumstances where the group members and instructor agree clearly that there was a significant lack of work performed. If one of your partners is not doing his/her share of the work, it is your responsibility to attempt to convince them to contribute. If this is not successful, notify Allison promptly. If we are not notified of a group problem until the last minute, there is not a good chance that a group problem can be resolved, nor different grades assigned.

Deliverables and Deadlines:

You will submit your work incrementally over the next three weeks. The work is divided into the following "**milestone**" deliverables. Each milestone can be submitted **up to 24 hours late for a -5% penalty, and will not be accepted more than 24 hours late.** Each phase is graded on **functionality ("external correctness") only**, aside from Milestone 4. If any group members have late days remaining, those group members will lose late days rather than points.

Milestone 1: Project Planning Due Tuesday, February 22, 11:59 PM (5% of grade)

A document containing the following information:

- Your group members names.
- A paragraph describing what you will be simulating
- A list of all of the classes you plan to implement and for each class all of their public member functions
- A drawing of your inheritance hierarchy

This should be submitted either as a `.txt` file or as a PDF.

Milestone 2: Beta Code Due Friday, March 4, 11:59 PM (25% of grade)

The first version of your code implementation will be called the "milestone" version. For this version we want to see that your group has completed a significant chunk of the work of the project, though we do not expect the simulation to be fully functional. We expect that by this date, we should be able to compile and run a version of your program successfully. When the program is run, we should be able to at least see text output of the initial contents of the grid. We should also be able to run some GUI code that at least opens up a GUI window and draws something. It is allowed for this milestone to contain some bugs as long as the general functionality can be run and tested.

Milestone 3: Project Presentation Due Wednesday, March 9, 2:50 PM, in class (5% of grade)

On the last day of lecture in class, we will perform short demos of all groups' projects. During these demos, one of your group members will run your simulation on his or her laptop while your group speaks briefly about your work and simulation and the features you chose to implement. **All members of your group must be present in lecture on the last day** of class and speak briefly about your project to receive credit for this milestone. Your project should be generally finished by this date so that it can be demoed successfully. But your source code does not have to be 100% complete for submission until the Friday of the last week of classes as documented below.

Milestone 4: Gallery GIF and Description Due Wednesday, March 9, 11:59 PM, in class (3% of grade)

You will submit an animated image of your project running as well as a paragraph describing what your simulation does. These will be posted on the course website on a final project gallery page.

See separate linked instructions for how to create an animated image (GIF) of your program.

Milestone 5: Final Code Due Friday, March 11, 11:59 PM (60% of grade)

You will submit the final version of your code and resources on the last day of classes for winter quarter. The final version will be graded more strictly than the other milestones and will be expected to meet all requirements for the project as described in this document. The correctness of your code will be graded by running your program and manipulating its graphical user interface, input files or constants. It will be very important to include good documentation about how to run your project. Exceptions and segfaults should not occur under normal usage. Your code should not produce any debug console output but you may output other information to the console if it adds value to your simulation.

Milestone 6: Group Evaluations Due Saturday, March 12, 11:59 PM (2% of grade)

After you've submitted your final code, you will also fill out a short survey about each of your project partners and what work you and the others did on the project. This will help us make sure that every group member contributed significantly to the project.