UPE Tutoring:

CS 32 Project 3 Hack

Sign-in https://bit.ly/39sW7vD

Slides link available upon sign-in

Quick Announcement - CS Town Hall

Date & Time: Wed, Feb 26th 6-8 pm (Week 8)
Location: Mong Auditorium, Eng VI

The CS Town Hall is an opportunity for all students in CS and related disciplines to interact with the UCLA CS Department and provide feedback, suggestions or discuss problems.

Please make your voice heard! Fill out this survey: https://forms.gle/JrE9HYXmwftrYB6NA

This year, we will also have a discussion about cheating in CS classes. Voice your opinion in this **completely anonymous** survey: https://forms.gle/KuRKjmfWqJZp8xix5

Food will be provided! RSVP Link: https://www.facebook.com/events/528498164437871/

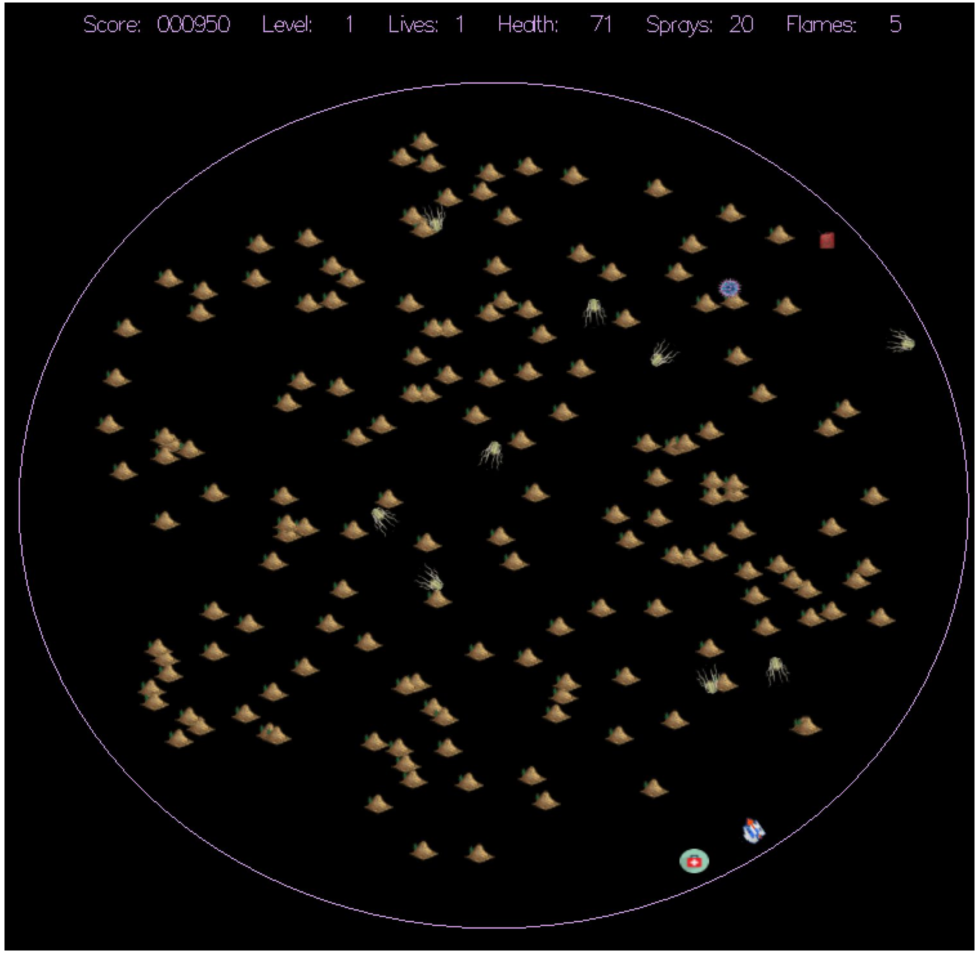
To ease your nerves... (tips and advice)

- The spec might look very long and intimidating, but it is actually very useful because:
  - Some parts sound redundant; this implies using inheritance / a common base class
  - Very detailed, so what you need to code is very clear; just translate English to code
- **START EARLY! DON'T PROCRASTINATE**
- Develop incrementally; code one feature at a time, then compile and test it
- Backup your code after each working feature gets implemented (github or Google Docs)
- **Don't procrastinate!** Catching up will be very stressful.
- **START EARLY!** Stop thinking about it, just DO IT!
- If you still have questions about game logic and implementation that the spec doesn't answer, refer to the example programs
- This project teaches Object-Oriented Programming, the most important skill you'll ever learn and the skill you'll use most if you're planning to work in industry!

Overview of Project

Kontagion: Brief Summary

- 2D game (that is very hard to play)
- Setting: Circular Petri dish / arena, nothing can be outside specified radius
- Goal: kill all bacteria that are spawned from bacterial pits on the level, go to next level
- Objects in the game:
  - Player: Socrates Nguyen
    - Can only move around perimeter of circle
    - 3 lives total
  - Projectiles: flame and disinfecting spray
  - Bacteria: 3 types that range from not aggressive to very aggressive
  - Goodies (power ups): 3 types that update Socrates' stats in a positive way
    - Harmful goodie: Fungus, deals 20 points of damage to Socrates
  - Other actors: Bacterial pits, Dirt piles, Food


Score: 000950 Level: 1 Lives: 1 Health: 71 Sprays: 20 Flames: 5

List of All Game Objects

Refer to page 21 of the spec

- Socrates
- Regular Salmonella bacteria
- Aggressive Salmonella bacteria
- Very aggressive E. coli bacteria
- Flame projectiles
- Spray projectiles
- Dirt piles
- Food
- Bacterial pits
- Restore health goodies
- Flame thrower goodies
- Extra life goodies
- Fungi

Actors

- Game Objects are called Actors
- Each ‘tick’ of the game, all of the Actors have to ‘doSomething’
    - They move around, cause damage, grant bonuses, die, etc.
- Each ‘tick’, the “dead” Actors should be removed
- A class called StudentWorld manages all of these ‘ticks’ and Actor interactions
- Project Goal: Implement all of the Actors and the StudentWorld

Getting Started

From the CS 32 website:

- Read the spec (54 pages! :D watch out for the occasionally funny bits)
- Download the skeleton code zip
- Play through the example games
- Do the Project 3 Warmup to get some experience with STL containers and iterators
- Re-read the spec as necessary
- Make sure you know how to compile and run your code

Project Structure

- Many files in the skeleton code, but we are only interested in some of them!
- Files of Interest:
    - Need to modify:
        - Actor.h/cpp
        - StudentWorld.h/cpp
    - Leave as is, but good to understand:
        - GameConstants.h
        - GameWorld.h/cpp
        - GraphObject.h
        - main.cpp (but you do need to set Assets string)

Project Structure: GraphObject Class

```
GraphObject(int imageID, double startX, double startY,
            int startDirection = 0, int depth = 0);
```

- Responsible for drawing all of the game objects at their (x, y) location
- Upon every instantiation of a GraphObject (including its derived classes), the provided code will display the appropriate image (according to its `imageID`) at (`startX`, `startY`)
- You MUST derive all of your game objects directly or indirectly from GraphObject
- Useful GraphObject functions (spec p. 22) for Part 1:
  - `double getX(); double getY();`
  - `void moveTo(double x, double y); void moveAngle(Direction angle, int units = 1);`
  - `void getPositionInThisDirection(Direction angle, int units, double& dx,double& dy);`
  - `int getDirection(); void setDirection(Direction d);`

Project Structure: GameWorld

- Controls the gameplay
- Keeps track of levels, lives, score
- Gets player input to move Socrates
- Plays sounds for various gameplay events
- Sets the game status text
- You MUST create StudentWorld, which is derived from GameWorld
  - Part 1: Implement at least private data members, `init()`, `move()`, `cleanup()`, constructor, destructor
- Useful GameWorld functions for Part 1 (spec p. 15): (you do need to use this one)
  - `bool getKey(int& value);`

Part 1

Due Week 7 Thursday
Feb 20, 11 pm

To do: (spec p. 50)

- Base class for all Actors (simplified)
- Dirt pile class
- Socrates class (simplified)
- StudentWorld class (simplified)

Part 1: Base Actor Class

- Create a base class from which all other game object classes will derive
- This base class must derive from GraphObject
- The constructor must initialize the object appropriately
- It must have a `doSomething()` function
- Add any public/private member functions and any private data members you need
  - What common functionality or attributes do all Actors have or need?
  - Which functions should be virtual? or pure virtual? Will this class ever be instantiated?
- Recommendation: Start with this class

Part 1: Dirt Pile Class

- Dirt piles can be derived from the base class
- Details (spec p. 27):
  - When first created:
    - Image ID: IID_DIRT
    - Location: Random, as specified in `StudentWorld::init()` section (p. 18)
    - Direction: 90 degrees
    - Depth: 1
    - Life: Initially alive; does not have hit points
  - During a tick (`doSomething()`)
    - Does nothing!
  - Other:
    - Blocks bacteria, spray, flames; dies from contact with spray or flames

Part 1: Socrates Class

- Socrates can be derived from the base class or from other derived classes
- Details (spec p. 25):
    - When first created:
        - Image ID: IID_PLAYER
        - Location: Positional angle of 180 degrees in Petri dish, so X = 0, Y = 128
        - Direction: 0 degrees (to the right)
        - Depth: 0
        - Life: Initially alive, with 100 hit points (health)
        - Weapon charges: 20 spray, 5 flame thrower

Part 1: Socrates Class (simplified `doSomething()`)

- Details (spec p. 25):
  - During a tick:
    - If Socrates is not alive, return immediately (not a specific Part 1 requirement)
    - Get user input; if player pressed a key, perform the desired action
      - If directional keys: move Socrates by 5 degrees in positional angle either clockwise or counterclockwise, and set the direction he is facing
      - Other keys (space to use spray, enter to use flamethrower): not Part 1 req
    - If player DID NOT press key: (not Part 1 req)
      - Replenish spray charges by 1 (if not already at maximum of 20 charges)

Part 1: Socrates Class (Object-Oriented Programming)

- But how can we check user input?
- How can Socrates get access to the `GameWorld::getKey(int& value)`?

- StudentWorld derives from GameWorld
- StudentWorld creates Socrates
- Is Socrates aware of StudentWorld? Is StudentWorld aware of Socrates?
- If they are aware of each other, who initiates an action to change the state of the other?
- **`THIS`** is quite an important dilemma in object-oriented programming
- Refer to the spec's Object Oriented Programming Tips (p. 45) for more info
- (Think back to Project 1, with the Arena and its Vampire and Player objects)

Part 1: StudentWorld Class (simplifed)

- Add private data members to keep track of all game objects (Actors and Socrates)
  - Part 1: will only be keeping track of dirt and Socrates for now
- Constructor: initializes data members
- init() (p. 16)
  - This method is called at the beginning of each new level
  - Allocates/initializes Socrates and all other Actors (for Part 1, just Dirt)
    - Refer to page 18 for details about how to initialize the Actors
- move() (p. 17-20)
  - Call `doSomething()` for every Actor (Socrates, Dirt, etc.)
  - Then delete dead Actors, add new Actors, update game status text (not Part 1)
- `cleanUp()` and Destructor (which can call `cleanUp()`) (p. 21)
  - Delete any remaining dynamically allocated game objects

How to keep track of your Actors

“It is **required** that you keep track of all of the actors (e.g., bacteria like aggressive salmonella, pits, foods, projectiles like spray, goodies, etc.) in a **single STL collection** such as a list, map or vector. (To do so, we recommend using a **container of pointers to the actors**). If you like, your *StudentWorld* object may keep a separate pointer to Socrates object rather than keeping a pointer to that object in the container with the other actor pointers; Socrates is the only actor pointer allowed to not be stored in the single actor container.” - The Spec, page 16

- What STL container do you want to use for your Actors?
    - Project 3 Warmup can guide you
    - So can Carey’s glorious slides about STL containers

Reminders about Iterator Shenanigans

- Deleting dead actors will require iterating through the STL container and only erasing a few of the items
- Thus, we need to use the return value of `erase()` which is an iterator to the next element after the deleted element
- These examples show how to erase all elements in a vector

BAD: `it` will be invalidated

```
vector<int> v{ 1 , 2 , 3 };
vector<int>::iterator it;
for (it = v.begin(); it != v.end(); it++)
    v.erase(it);
```

BETTER: `erase()` returns a valid iterator which is basically `it++`

```
vector<int> v{ 1 , 2 , 3 };
vector<int>::iterator it;
for (it = v.begin(); it != v.end(); ) // notice: no it++
    it = v.erase(it);
```

- What happens if we only want to erase some elements from the STL container?

Reminders about Iterator Shenanigans

- When erasing only some items, make sure not to accidentally skip elements!
- These examples are with a STL list, but for all STL containers we should use the return value of `erase()`

BAD: `it++` will cause elements to be skipped

```
void eraseOnlyZeroes(list<int>& x) {
    list<int>::iterator it = x.begin();
    for ( ; it != x.end(); it++) {
      if (*it == 0)
          it = x.erase(it);
    }
}
```

BETTER: only increment `it` when we DO NOT erase

```
void eraseOnlyZeroes(list<int>& x) {
    list<int>::iterator it = x.begin();
    for ( ; it != x.end(); ) {  // notice: no it++
      if (*it == 0)
          it = x.erase(it);
      else
          it++;
    }
}
```

Part 2

Due Week 8 Thursday
Feb 27, 11 pm

- Implement everything in the spec to create a fully working game
- Files you need to complete:
  - Actor.h/cpp
  - StudentWorld.h/cpp
- And a report
  - Don't overdo the detail
- After turning in Part 1, the CS 32 website will update with resources discussing possible designs
- Good luck and have fun!

Object-Oriented Design Tips

Polymorphism and Inheritance

- Although it is a long spec, a lot of it is repetitive
- Take the “subtle” hint!! - Use polymorphism and inheritance!

FLIP BACK AND FORTH BETWEEN THE NEXT TWO SLIDES TO SEE FOR YOURSELF

copied straight from the spec

What a Regular Salmonella Must Do During a Tick

Each time a regular salmonella is asked to do something (during a tick):

1. The regular salmonella must check to see if it is currently alive. If not, then its *doSomething()* method must return immediately – none of the following steps should be performed.
2. The regular salmonella must check to see if it overlaps[11] with Socrates. If so it will:
    a. Tell Socrates that he has received 1 point of damage.
    b. Then skip to step 5.
3. The regular salmonella will see if it's eaten a total of 3 food since it last divided or was born. If so:
    a. It will compute a *newx* coordinate, equal to its own *x* coordinate:
        i. Plus SPRITE_RADIUS if its *x* coordinate is < VIEW_WIDTH/2
        ii. Minus SPRITE_RADIUS if its *x* coordinate is > VIEW_WIDTH/2

Page 37

copied straight from the spec

<u>What an E. coli Must Do During a Tick</u>

Each time an E. coli is asked to do something (during a tick):

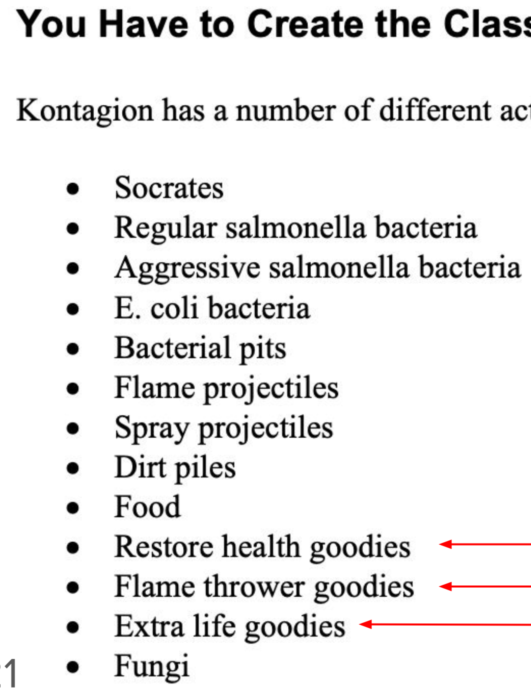
1. The E. coli must check to see if it is currently alive. If not, then its *doSomething()* method must return immediately – none of the following steps should be performed.
2. The E. coli must check to see if it overlaps[17] with Socrates. If so it will:
    a. Tell Socrates that he has received 4 points of damage.
    b. Then skip to step 5.
3. The E. coli will see if it's eaten a total of 3 food since it last divided or was born. If so:
    a. It will compute a *newx* coordinate, equal to its own *x* coordinate:
        i. Plus SPRITE_RADIUS if its *x* coordinate is < VIEW_WIDTH/2
        ii. Minus SPRITE_RADIUS if its *x* coordinate is > VIEW_WIDTH/2

Page 43

Polymorphism and Inheritance

- Plan out your classes and inheritance structure before starting to code!
    - Draw a potential class diagram that makes sense
- Very easy to implement Part 1 with a poor code structure
    - Resist the temptation.
- Plan ahead! Make your life easier when it's time to do Part 2!

Polymorphism and Inheritance: From the Spec



You Have to Create the Classes for All Actors

Kontagion has a number of different actors, including:

- Socrates
- Regular salmonella bacteria
- Aggressive salmonella bacteria
- E. coli bacteria
- Bacterial pits
- Flame projectiles
- Spray projectiles
- Dirt piles
- Food
- Restore health goodies
- Flame thrower goodies
- Extra life goodies
- Fungi

wow maybe i should make a Goodie class and derive from it

Page 21

Miscellaneous Topics

that may be helpful for beyond Part 1

Generate Random Numbers

In GameConstants.h:

```
// Return a uniformly distributed random int
// from min to max, inclusive

int randInt(int min, int max)
{
    ...
    implementation provided for you already
    ...
    return a random number;
}
```

Stringstreams (for game status text)

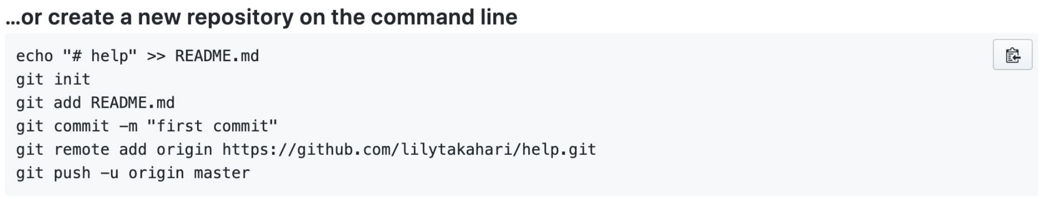
```
#include <iostream>
#include <sstream>
#include <iomanip>

ostringstream oss;
int n = 123;

oss << setw(5) << k << endl;
oss.fill('*'); // change fill from ' ' to '*'
oss << setw(6) << k << endl;
string s = oss.str(); // s contains "  123\n***123\n"
```

Github: Backing Up Code

- You could use Github as a fancy Google Docs by copying and pasting all your files
- Or you could create a local git repository and push it to Github by typing a few lines into Terminal/PuTTY/command line

Steps:

1. Download Git: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
    a. For Mac: You most likely already have it installed if you have Xcode
2. Create a new repository on your Github account.
3. You will probably see a screen with info like this:



…or create a new repository on the command line

```
echo "# help" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/lilytakahari/help.git
git push -u origin master
```

Github: Backing Up Code

4. In Terminal/PuTTY, cd (change directory) to the folder where all your source code is

   e.g. `cd ~/Documents/Winter_2020/Kontagion/Kontagion`
5. Initialize a new git repository: `git init`
6. Add all the files in the folder to the repository:

   ```
   git add .
   ```
7. Commit them to the repo. MUST include a message/string after -m:

   ```
   git commit -m "first commit"
   ```
8. From the screen on GitHub, enter the command that starts with git remote add:

   e.g. `git remote add origin https://github.com/lilytakahari/help.git`
9. Push your code to the web:

   ```
   git push -u origin master
   ```

Repeat steps 6, 7, and 9 everytime you want to backup your code to Github.

Useful Pages for Part 1

- 48: Don’t know how or where to start? Read this!
- 6: Game Details
- 44: Object-Oriented Programming Tips
- 50: What to Turn In, Part #1
- 14: StudentWorld description
- 21: Actors description
- 25: Socrates
- 28: Dirt Pile
- 9: Determining Object Overlap
- 49: Building the Game

Good luck!

Sign-in	https://bit.ly/39sW7vD
Slides	https://bit.ly/3busBHz
Practice	https://github.com/uclaupe-tutoring/practice-problems/wiki

Questions? Need more help?

- Come up and ask us! We'll try our best.
- UPE offers daily computer science tutoring:
  - Location: ACM/UPE Clubhouse (Boelter 2763)
  - Schedule: https://upe.seas.ucla.edu/tutoring/
- You can also post on the Facebook event page.