UPE Tutoring:

CS 32 Project 3 Hack

Sign-in	https://goo.gl/MG9dVH
Slides	https://goo.gl/qkhQuF

Overview of Project and Part 1

Overview

- Implement a game called NachenBlaster
- Side Scrolling Shooter Game
- Brief Summary of the Game:
    - Your ship must shoot cabbages at alien ships to destroy them
    - 3 types of alien ships - each type has different behavior
    - Defeated aliens sometimes drop power ups that help you
    - Destroy enough aliens to advance to the next level
    - Lose all of your health points and you die (3 lives per game)
    - There’s a nice background of stars that constantly move to the left

Overview

Various Game Objects:

- Star
- NachenBlaster
- Smallgon
- Smoregon
- Snagglegon
- Cabbage
- Turnip
- Torpedo (Projectile)
- ExtraLife
- Repair
- Torpedo (Goodie)
- Explosion

Overview

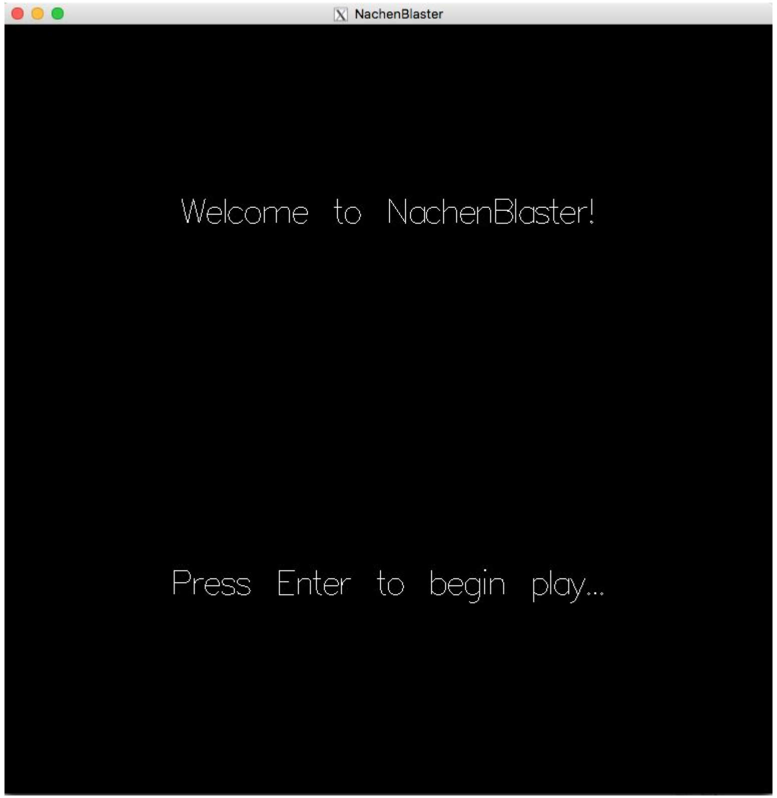
- Game Objects are called Actors
- Each ‘tick’ of the game, all of the Actors have to ‘doSomething’
  - They move around, cause damage, grant bonuses, etc.
- Each ‘tick’, the “dead” Actors should be removed and any new Actors should be created
- A class called StudentWorld manages all of these ‘ticks’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

Overview

- ~~Full spec not available yet :(~~
- ~~Enough of the spec has been released to complete Part 1~~
- Full spec now available!! :D (51 pages)

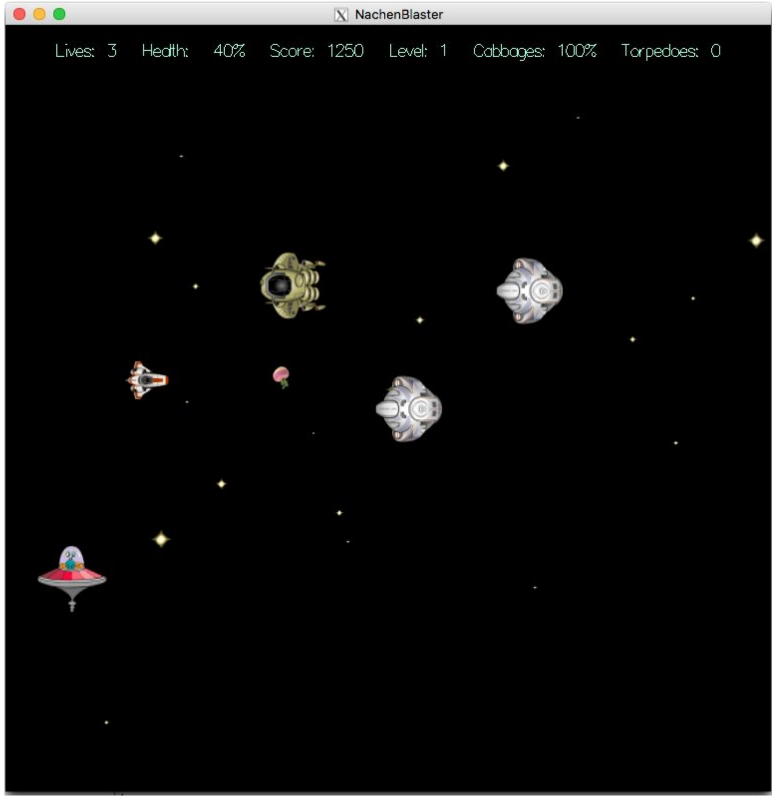
- Examples of the complete game are available on the CS32 website
- Skeleton Code is also available on the website

Overview


NachenBlaster

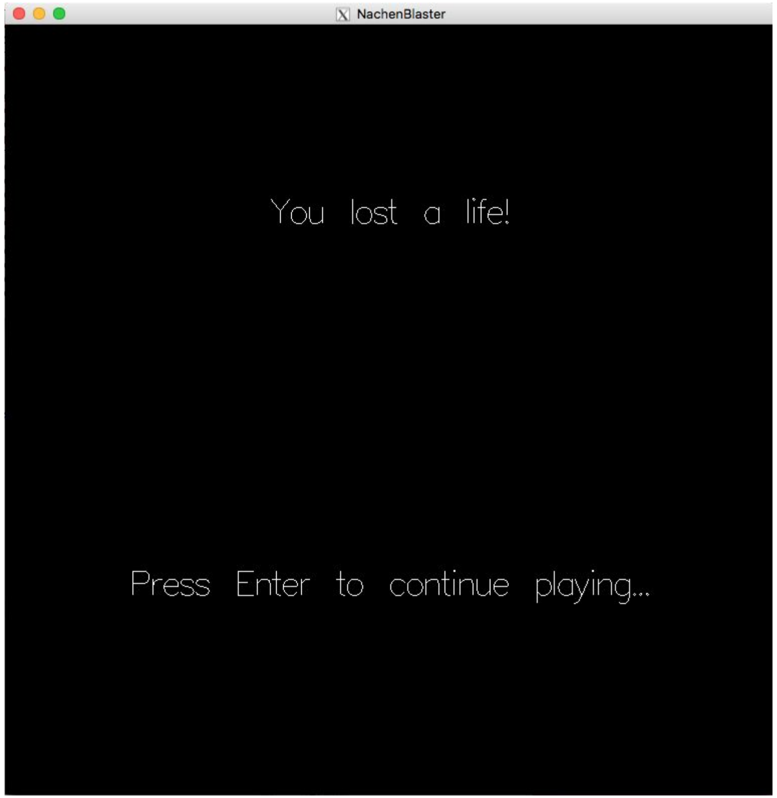
Welcome to NachenBlaster!

Press Enter to begin play...


NachenBlaster

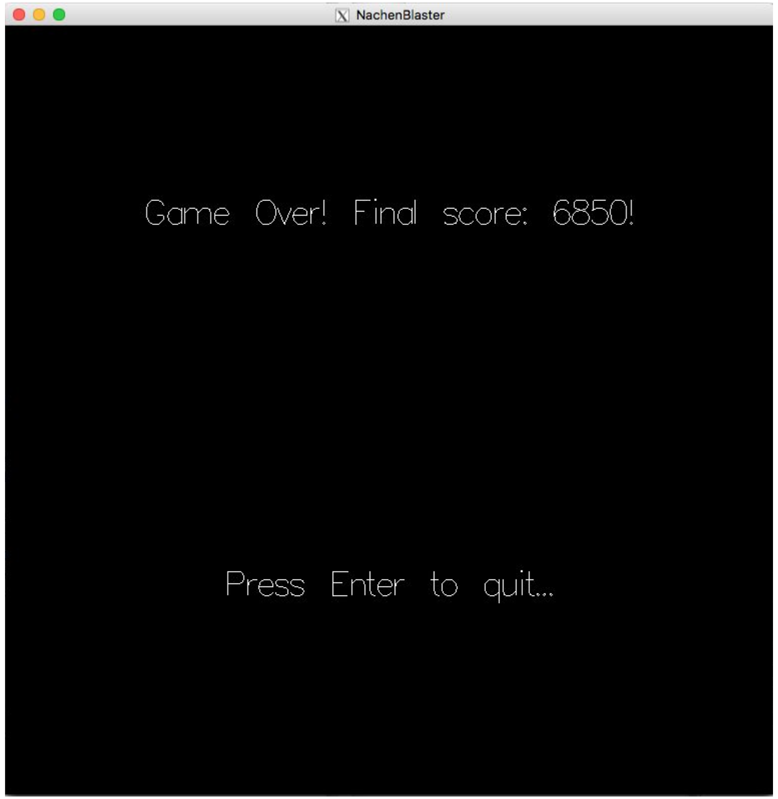
Lives: 3 Health: 40% Score: 1250 Level: 1 Cabbages: 100% Torpedoes: 0

Overview


NachenBlaster

You lost a life!

Press Enter to continue playing...


NachenBlaster

Game Over! Final score: 6850!

Press Enter to quit...

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:
    - GameConstants.h
    - GameWorld.h/cpp
    - GraphObject.h
    - main.cpp (but you do need to set assetDirectory variable)

Project Structure: GraphObject Class

- Responsible for drawing all of the objects
- Every instantiation of GraphObject (including derived classes) will draw something on the space field according to the Image ID
- Draw the image at location X and Y
- Useful functions for Part 1: (you may or may not need to use each one)
  - getX()
  - getY()
  - moveTo(double x, double y)
  - setSize(double size)

Project Structure: GameWorld

- Controls the gameplay
- Keeps track of lives, score, level
- Gets player input to move NachenBlaster
- Play sounds for various gameplay events
- StudentWorld is derived from GameWorld
- Useful Functions for Part 1: (you do need to use this one)
  - getKey(int& ch);

Part 1 To Do

- Base Class for all Actors
- Star class
- NachenBlaster class (simplified)
- StudentWorld class (simplified)

Part 1: Base Actor Class

- Create a base class which all other classes will derive
- This base class derives from GraphObject
- It should have a doSomething() function
  - Which functions should be non pure virtual or pure virtual? Will this class ever be instantiated?
- Add any public/private member functions and any private data members you need
- Recommendation: Start with this class

Part 1: Star Class

- Stars are derived from the base class
- Details:
  - Image ID: IID_STAR
  - Location: random x and y if created at beginning of a level, only random y if created during a level
  - Direction: 0
  - Size: random from 0.05 to .50
  - Depth: 3

Part 1: Star Class

- During a tick (doSomething method)
    - Move 1 pixel to the left (GraphObject's moveTo method!)
    - If it moves off screen, it needs to tell StudentWorld to delete it
- At the beginning of a level, there are 30 stars randomly placed
- Each tick, there is a 1/15 chance a new star is created

Part 1: NachenBlaster Class

- NachenBlaster is derived from the base class (or from another class that derives from the base class)
- Details:
  - Image ID: IID_NACHENBLASTER
  - Location: starts at x=0, y=128
  - Direction: 0
  - Size: 1.0
  - Depth: 0
  - 50 health points
  - 30 cabbage energy points

Part 1: NachenBlaster Class

- During a tick (doSomething method)
  - Check to see if NachenBlaster is alive
    - do nothing if not alive
  - Check user input and perform action
    - getWorld->getKey(ch);
    - Move in a direction within bounds of space field
    - Fire a cabbage or torpedo (not needed for Part 1)

Part 1: StudentWorld Class

- Add private data members to store the NachenBlaster and Stars
- Constructor - initialize data members
- init()
  - This method is called at the beginning of each level
  - Create the NachenBlaster and the starting 30 stars
- move()
  - Call the doSomething() for every Actor (NachenBlaster, Stars, aliens, etc.)
  - Delete any ‘dead’ Actors
  - Possibly create a new star on a 1/15 chance
- cleanUp() and Destructor
  - Delete any remaining dynamically allocated game objects

Miscellaneous Topics and Tips

Random Numbers (Probability)

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;
}
```

Random Numbers (Probability)

```
int n;
n = randInt(0,9);  // n is now a number from [0 , 9]

if (n == 0); // 1/10 chance to be true: 0,1,2,3,4,5,6,7,8,9
if (n < 5); // 1/2 chance to be true: 0,1,2,3,4,5,6,7,8,9

n = randInt(10, 19);  // n is now from [10 , 19]

double n_d;
n_d = n / 100.0;  // n_d is now from [.1 , .19], step = .01
```

Stringstreams

```
#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"
```

Miscellaneous Tips

- Don’t be intimidated by the length of the spec!!
- It’s purposefully written to be a very long spec
  - Notice the similarities between game objects
  - Take the “subtle” hint!! - Use polymorphism and inheritance!

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

From the Spec

What a Smallgon Must Do During a Tick

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

1. The Smallgon 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 Smallgon must determine if it's flown off of the left side of the screen (its x coordinate is less than 0), and if so, mark itself as dead so that the StudentWorld class can remove it from the space field. If so, it must do nothing more during the current tick.
3. The Smallgon must check to see if it has collided with a NachenBlaster-fired projectile or the NachenBlaster ship itself (see the Euclidian collision detection approach on page 10). If so, it must follow the steps outlined in the Collision section below.

Page 34

From the Spec

What a Smoregon Must Do During a Tick

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

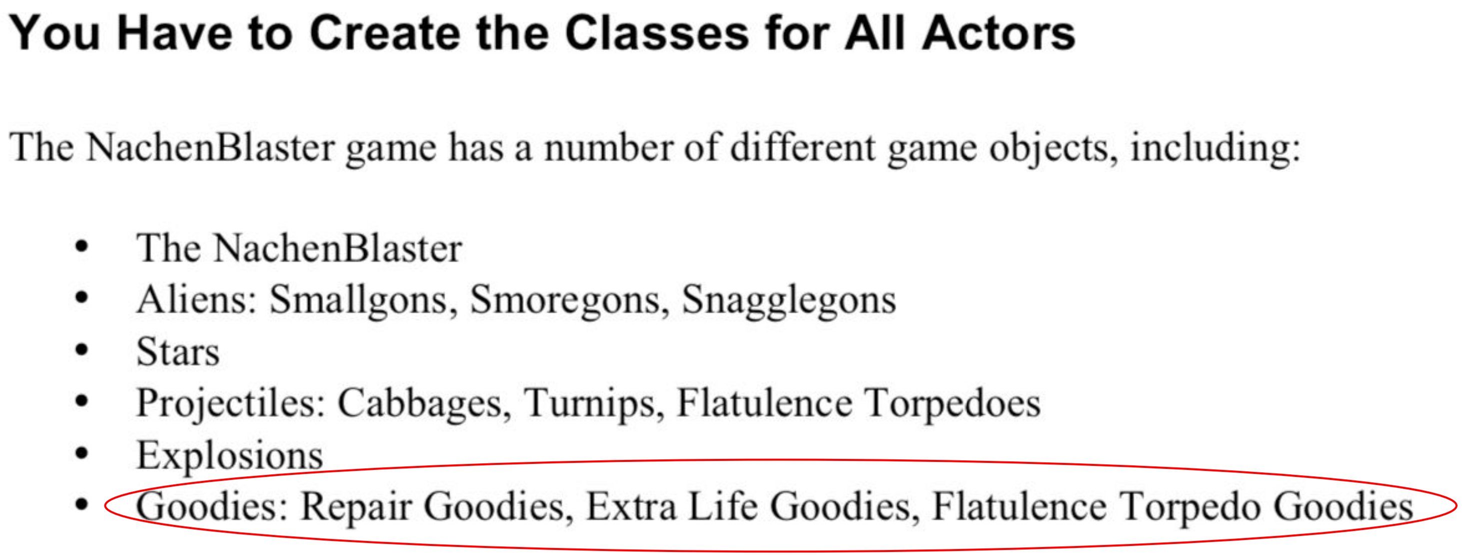
1. The Smoregon must check to see if it is currently alive. If not, then its doSomething() method must return immediately – none of the following steps should be performed.
2. The Smoregon must determine if it's flown off of the left side of the screen (its x coordinate is less than 0), and if so, mark itself as dead so that the StudentWorld class can remove it from the space field. If so, it must do nothing more during the current tick.
3. The Smoregon must check to see if it has collided with a NachenBlaster-fired projectile or the NachenBlaster ship itself (see the Euclidian collision detection approach on page 10). If so, it must follow the steps outlined in the Collision section below.

Page 36

Miscellaneous Tips

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

From the Spec



You Have to Create the Classes for All Actors

The NachenBlaster game has a number of different game objects, including:

- The NachenBlaster
- Aliens: Smallgons, Smoregons, Snagglegons
- Stars
- Projectiles: Cabbages, Turnips, Flatulence Torpedoes
- Explosions
- Goodies: Repair Goodies, Extra Life Goodies, Flatulence Torpedo Goodies

Page 20

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

Miscellaneous Tips

- Remember to use STL data structures
- StudentWorld needs to keep track of many things
- Use vectors, lists, stacks, queues, sets, maps, etc.
- Plan out what data structures you need to keep track of all game objects

Miscellaneous Tips

- Testing random events can be tricky
- Change the probability!
  - Ex. A new star only has a 1/15 chance to be created, but we want to make sure stars are being created correctly
  - (randInt(1, 15) == 1) ➔ (randInt(1, 15) > 1)
- Remember to change it back before you submit!

Miscellaneous Tips

Useful Pages in the Spec for Completing Part 1:

- Game Details (how the game should look and play) - Page 6
- StudentWorld Description - Page 13
- NachenBlaster Description - Page 23
- Star Description - Page 26
- Object Oriented Programming Tips - Page 41
- What to Turn in - Page 47

Miscellaneous Tips

- **Start early!!**
- Compile your code often so you’re never overwhelmed by bugs
- Make backups of your code
- Tackle each feature one at a time
- As thorough as the spec is, you still may have questions about implementation and game logic. The example programs on the website are great references for what should be done
- **Don’t procrastinate!! This is a long project!!**
- **START EARLY!!**

Good luck!

Sign-in	https://goo.gl/MG9dVH
Slides	https://goo.gl/qkhQuF
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.