LECTURE 2
Announcements
Tac 1 is done!

• Initial feedback for Tac1
  – Value noise is looking good! Nice maps!
  – Viewports are important – fix them now!
  – Anti-aliasing can destroy your framerate in some scenarios – make it toggleable!
  – Use Vec’s! We will give INCOMPLETES!

• Next week your game will really start to take form, and you’ll pick a genre!
  – TBS or RTS!
Don’t Forget Tic

• Retries should all be in
  – Please send us an email after you hand in any retry

• A few more tips for the future…
  – Watch out for edge cases
  – Plan out game/engine separation before you start
You’re Almost There

You are here

Number of hours

Tic  Tac I  Tac II  Tac III  Tou I  Tou II  M I  M II  M III
Announcements

QUESTIONS?
LECTURE 2
Pathfinding
Pathfinding

MOTIVATION
Why is pathfinding important?

• NPCs need to navigate an environment that has obstructions

• Goal: find minimum cost path from A to B
  – Cost includes factors such as distance, terrain, position of enemies.

• Typically uses a graph to represent navigable states.
DIJKSTRA’S ALGORITHM

Pathfinding
Dijkstra’s

• Basic idea:
  – Process nodes in order of shortest distance from start
  – To process a node, update cost to each neighbor and add to PriorityQueue, then never process this node again
  – Each node keeps track of shortest distance and pointer to previous node
  – When it’s time to process the end node, you’re done
Why Dijkstra’s can be gross
Pathfinding

A*
General idea

• Dijkstra’s assumes it’s impossible to predict cost
  – This is overly pessimistic
• In pathfinding, we at least know the general direction we want to go
• A* is a graph traversal algorithm that takes advantage of this
How does it work?

• Uses a “heuristic” to guess the cost from any given node to the destination node
  – Heuristic passed by the caller
• In addition to tracking distance from start, track heuristic value for each node
  – Prioritize in PriorityQueue based on distance+heuristic
• This can be as simple as the Euclidean distance between the given and destination node, but also try to take other factors
  – Get creative – better heuristics help A* run faster!
What could possibly go wrong?

- Overestimating the cost from start to finish can result in sub-optimal results
- Trying to find a path from S to G
  - Note that G is a neighbor of S, but the shortest path is actually S,A,C,G (4 vs. 12)!
  - What if the heuristic estimate of A to G was 25?
  - Explore G first -> done!
- Avoids expanding the optimal path because the heuristic indicates “no, no, trust me, A is SO much further from G”
  - Most distance-based heuristics should be fine
Pathfinding

QUESTIONS?
LECTURE 2
Graphics II
WHAT’S A SPRITE?
THIS IS A SPRITE
Sprites as Bitmap Data

• “Raster” graphics
• Pre-constructed images dynamically placed on the screen
• Designed to represent one type of object in a game
  – Objects may reference different sprites depending on state
Sprites as Animation Data

- Sprites as a filmstrip
- Designed to represent frame-by-frame snapshots of a single game object
- Standardized padding, size, and spacing allows for easy drawing
Typical Sprite File Format

- Multiple frames per file
- Multiple directions per file
- Multiple objects per file
- Additional information often (but not always) in config files:
  - Padding
  - Bounding boxes
  - Locations of a particular object’s sprites
- Standard sprites:
  /course/cs195n/support/tac/sprites.png
Formatting “Standards”
Additional Sprite Info

• Bounding boxes for...
  – Physics
  – Entity logic (collides, contains, etc)
  – Screen area distribution

• Other types of information?

• Optional for now
  – But think how you might need this in Tou or M
IMPLEMENTING SPRITES
Sprite Loading

• You should only load a sprite sheet once
  – Each object using the sprite maintains a reference to it
• Consider making a Resource class which loads in sprite sheets
  – Dynamically load in sprites when you need them
  – Handle sprite indexing for animations
  – Generalizable to other assets like maps, sounds, text, etc…
Drawing Sprites

• About `Graphics2D.drawImage()` ...

• Write a custom sprite drawing routine, and pick the one that works best for you
  – If you’re using affine transforms, there’s a special method available

• Your drawing routine should handle different padding and formats
Drawing Frames

• Draw rectangular chunks from sprite sheet to the canvas
  – Calculate based on elapsed time, frame width and height

• Don’t cache scaled images
  – Scaling up every time is worth the space/speed tradeoff

• Remember to draw from your single sprite sheet reference
Animating on Ticks

- Animate by drawing frames like a flipbook: one after another, then reset
- Update frame state on tick, draw correct frame on draw
  - Draw is read-only
- Use modulo (%) operator to start over
LECTURE 2
Tips for Tac 2
Unit movement

- Tiles best stored in an array (discrete indices)
- But game space is continuous!
- Define tile \( x,y \) to take up space \([x, x+1) \) and \([y, y+1)\)
- Move unit centers with \( \text{Vec2f}.\text{lerpTo()} \)
RTS - Unit exclusion

- Unit sitting on tile is obvious
- But what about while moving?
  - When is it no longer on the first tile?
  - When does it officially reach the second tile?
  - Can it briefly monopolize two tiles at once?
Removing Units

• Beware the ConcurrentModificationException!
  - Doesn’t actually have anything to do with threads
• Consider a removal queue
  - This can be generalized to multiple phases of ticks
• Consider iterating over a copy of the game data
Sprites

• You’ll need to have sprites in your game to make it pretty!

• We have tank sprites:
  – /course/cs195n/support/sprites.png

• More random sprites:
  – http://jessefreeman.com/category/game-artwork/
Tips for Tac I

JAVA TIP OF THE WEEK
Generics are cool!

• You’ve used generics before... but have you ever written them?

• It’s as easy as:

```java
public class SimpleContainer<T> {
    private T object;
    public void setObject(T ob) { object = ob; }
    public T getObject() { return object; }
}
```
Generics are cool!

• Can use `extends` and `super` to bound the type

```java
public class AnimalHouse<A extends Animal> {
    private A animal;
    public void houseAnimal(A a) { animal = a; }
    public void feedAnimal() { animal.eat(); }
}
```

```java
AnimalHouse<Dog> kennel; // okay
AnimalHouse<Rock> mountain; // compile error
```
Want to know more?

- *Effective Java* by Joshua Bloch has an excellent chapter on generics
- Gives examples of where advanced generics are useful
- Can be found in the back of the SunLab
Tips for Tac 2

QUESTIONS?
What is a game designer?

- A game designer creates the experience and the “feel,” and is not just a programmer.
- The game is not by itself the experience, just the delivery system for your ideas.
Sub-disciplines of game design

- World design
  - Backstory, setting, theme
- System design
  - Game rules (and math)
- UI design
  - Controls, menus, overlays
- Level design
  - Actual playable environments
The sadly wonderful truth...

- A game designer will never experience what he or she creates
- It is the player that ultimately interacts with the creation
- Their experience cannot be shared
- Listening is fundamental to designers
Compared to other media

- Designers of movies, books, and plays are creating a linear experience
- Interactivity is the defining feature of video games as a medium of entertainment
- “A game is a problem-solving activity approached with a playful attitude.” – Jesse Schell
Jesse Schell’s Four Elements

More visible

Aesthetics

Mechanics

Story

Technology

Less visible
MDA Framework
Where to begin?

• Approach from the player’s perspective
  – What aesthetics do you want your game to have?
  – What do you want your players to feel?

• Create a basic idea that encapsulates those aesthetics

• Come up with dynamics that evoke the aesthetics

• First and foremost: know your audience!
  – Sunlab users? Competitive MOBA veterans? Kids?
Bartle’s Taxonomy of Player Types

- Killers
- Achievers
- Socializers
- Explorers

Axes:
- Acting
- Interacting
- Players
- World
How to become a better designer?

• At Brown...
  – Make games, play games, come to BRGD

• Play lots of games!

• More specifically, play games like a designer
  – Extra Credits Season 2, Episode 2
Case Study: Super Mario Bros.
Building an aesthetic in Tac

Here’s one way you could build an aesthetic with A* in your Tac:

<table>
<thead>
<tr>
<th>Mechanics</th>
<th>Dynamics</th>
<th>Aesthetics</th>
</tr>
</thead>
<tbody>
<tr>
<td>• When a tile is clicked, then all selected units will path-find using A* to that tile</td>
<td>• Moving a mass of units to a particular location</td>
<td>• Player feels empowered by managing and controlling a swarm</td>
</tr>
<tr>
<td>• When an enemy is clicked, then all selected units will attack it</td>
<td>• Attacking something with a mass of units</td>
<td></td>
</tr>
<tr>
<td>• When the mouse is pressed and dragged, then all units inside the box become selected</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Things to See

• Extra Credits: Playing Like a Designer
  – https://www.youtube.com/watch?v=_HmtmoGwpZc

• Gamasutra - articles and news about game development
  – http://www.gamasutra.com

• MDA: A Formal Approach to Game Design
  – http://www.cs.northwestern.edu/~hunicke/MDA.pdf
Tac1 Playtesting

Yay!