Tou has been released!

- Shoot-’em-up, heavy use of collision detection
- Much more open-ended than previous projects
- Easier than previous projects if you work smart
  - Should help those of you combating the snowball of death
Behind on Tac?

• Don’t worry! You’re not alone. 😊

• Incomplete on Tac2?
  – Use an extra retry this week!

• Incomplete on Tac3?
  – Do Tou1 first, then try to catch up.
  – If you don’t catch up, it’s ok! You can still get an A!
Engine Requirements

• If a requirement is listed as an engine requirement, then the implementation should be in the engine
  – Otherwise you will get an incomplete
• There is more flexibility about game requirements
LECTURE 4
Game Worlds
Game Worlds

MOTIVATION
How you probably did Tac

Viewport

GameScreen

Application

FrontEnd

Tile[]

Unit

Bullet

AIUnit
How does this scale?

• Imagine implementing four-player split-screen:

Viewports

Normal UI element
Solution: separate world+screen
RESPONSIBILITIES
What does a world do?

- Represents all game objects ("entities") in a single space
  - "Owns" game coord system
- Centralizes entity management
  - Maintains list, add/remove via queues or iterating over copy
  - Passes through ticks and draws to entities
- Handles global logic
  - Anything beyond the scope of a single entity
  - E.g. providing collision detection callbacks to entities
Entities

• Single logical “object” within the game
  – Stores all state relevant to itself, e.g. draw shape, collision shape, HP, attack cooldown...
  – Should always know what World it’s in

• Lightweight superclass
  – High-level subclasses in engine, low-level in game

• Receives events from World
  – More than just tick+draw!
Entity vs Shapes

- Shapes are not Entities
- Entities are not Shapes
- No inheritance between them
- Entities should contain Shape(s)
Entity responsibilities

- May draw, may not
  - Spawners, timers, force fields
- May use ticks, may not
  - Static environment objects, background…
- Most will probably do both though
  - Player, items/pickups,
- Player input?
  - Might be better handled by World, your call
- Effects on other entities?
  - Might be better handled by World, your call
Multi-pass logic

• Ticking and drawing entities in the wrong order leads to undesirable behavior
  – Drawing background over everything else
  – Entities removing themselves during collision detection
• World can selectively update state in order
  – E.g. tick all entities so they update position, *then* check for collisions
  – Could have a way for specifying draw order
Game Worlds

QUESTIONS?
Collisions have consequences

- Object collisions are extremely important in everyday life
- Extremely important in games too
  - Games that use collision detection vastly outnumber those that don’t
What do we want to collide?

- Points
- Circles
- Axis-Aligned Boxes
- Convex polygons
  - Covered next week
- Compound shapes
- Other shapes
  - Not covered

\[ P_1, P_2, P_3, P_4, P_5 \]

\[ \text{dim} = (w, h) \]

\[ r \]

\[ \text{min}_x, \text{max}_x, \text{min}_y, \text{max}_y \]
Collision Detection | DETECTION ALGORITHMS
Point-Circle

• Check if the distance between the point and the center is less than or equal to the radius

\[ \|P - C\|^2 \leq r^2 \]
Circle-Circle

• Check if the distance between the two centers is less than or equal to the sum of the radii

\[ \|C_2 - C_1\|^2 \leq (r_1 + r_2)^2 \]
Point-AAB

- Check if point is within range on each axis
- $\min_x \leq P_x \leq \max_x$ AND $\min_y \leq P_y \leq \max_y$
AAB-AAB

• Ensure overlap on each axis:
  - $A_{x\text{min}} \leq B_{x\text{max}} \ \text{AND} \ \ A_{x\text{max}} \geq B_{x\text{min}}$
  - $A_{y\text{min}} \leq B_{y\text{max}} \ \text{AND} \ \ A_{y\text{max}} \geq B_{y\text{min}}$
Circle-AAB

- Check if closest point to circle on AAB is in circle
  - Closest point: clamp $C.x$, $C.y$ to $[\text{min.x, max.x}]$, $[\text{min.y, max.y}]$
  - Then just do point-circle collision
Compound-anything

• Compound shape checks against sub-shapes
  — Only compound shapes should ever need to iterate over children
Collision Detection | QUESTIONS?
LECTURE 4
Tips for Tou I
Write a collision debugger

- Collision math is tricky
- You will make mistakes
- Test your collision code before even putting it in a game
- This is required
Watch out for key “ghosting”

- Due to the way keyboards are built, sometimes keys don’t work while others are being held down
  - In many cases, as few as 3 can be held at a time
- Hardware issue (can’t fix it), you must design around it
  - E.g. use mouse for some actions
- More info, interactive demo: http://www.microsoft.com/appliedsciences/antighostingexplained.mspx
Double dispatch

• If you have a Circle and an AAB but only know that they’re Shapes, how do you determine which method to call?

```java
void testCollide() {
    Shape s = new Circle();
    Shape s2 = new AAB();
    s.collides(s2);
}

abstract class Shape {
    collides(Circle c);
    collides(AAB aab);
    collides(Comp m);
    collides(Shape o);
}

boolean collides(Shape o) {
    if (o instanceof Circle) {
        return collides((Circle) o);
    } else if (o instanceof AAB) {
        return collides((AAB) o);
    } else if (o instanceof Comp) {
        return collides((Comp) o);
    } else {
        throw new IllegalArgumentException();
    }
}
```
interface Shape {
    collides(Shape o);
    collidesCircle(Circle c);
    collidesAAB(AAB aab);
    collidesComp(Comp m);
}

public class Circle implements Shape {
    collides(Shape o) {
        return o.collidesCircle(this);
    }
    collidesCircle(Circle c) { /*code*/ }
    collidesAAB(AAB aab) { /*code*/ }
    collidesComp(Comp m) { /*code*/ }
}

public class AAB implements Shape {
    collides(Shape o) {
        return o.collidesAAB(this);
    }
    collidesCircle(Circle c) { /*code*/ }
    collidesAAB(AAB aab) { /*code*/ }
    collidesComp(Comp m) { /*code*/ }
}

public class Comp implements Shape {
    collides(Shape o) {
        return o.collidesComp(this);
    }
    collidesCircle(Circle c) { /*code*/ }
    collidesAAB(AAB aab) { /*code*/ }
    collidesComp(Comp m) { /*code*/ }
}
Variable arguments!

- Wouldn’t it be nice if instead of this:

```java
List<Shape> l = new ArrayList<Shape>();
l.add(new Circle());
l.add(new AAB());
l.add(new AAB());
Compound c = new Compound(l);
```

- Or this:

```java
Compound c = new Compound(new Shape[] {new Circle(), new AAB(), new AAB()});
```

- You could do this?

```java
Compound c = new Compound(new Circle(), new AAB(), new AAB());
```
Well you can!

• The type of the last argument of a constructor/method may end in ‘…’ to indicate “any number of arguments”

```java
public Compound(Color c, Shape... shapes)
```

• The “variable” argument will actually be an array of arguments (it’s really just wrapping array creation):

```java
public Compound(Color c, Shape... shapes) {
    for (int i=0; i < shapes.length; ++i) {
        add(shapes[i]);
    }
}
```
Passing arrays as varargs

• Since it's really just an array, it's possible to just pass in an array if you already have one:

```java
static void printAllOnALine(Object... objs) {
    for (Object o : objs) System.out.print("("+o+"") ");
    System.out.println();
}

public static void main(String[] args) {
    printAllOnALine(args);
    List<?> ents = new World().getEntities();
    printAllOnALine(list.toArray());
}
```
Juice it or lose it (Video)

https://www.youtube.com/watch?v=Fy0aCDmgnxg

Demo Available at: http://grapefrukt.com/f/games/juicy-breakout/
A warning...

- You should only add juice to your game in one situation: when your engine and game requirements are completed!
- Exception: particles for the final project
Juice in 195n

Basic juice:
• Color
• Mouse hover effects
• Sprites
• Sound*

(Less programming effort)

Advanced juice:
• Tweening
• Screen shake
• Screen freeze
• Particles*

(More programming effort)

*options for your final project
Feedback Loops

• The player wants to know
  – When damage is done
  – If things collide (especially projectiles)
  – If an action triggered or failed

• Good place to use Juice
QUESTIONS?
GAME DESIGN TIPS FOR TOU
General gameplay

- Two standard schemes for shoot ‘em ups:
  - Player can only shoot in one direction and enemies approach from that direction
  - Player can aim freely and move freely
Major design decisions

• How will the player aim at enemies?
  – With the mouse? By lining up with enemies?

• How freely can the player move?
  – One or two axes of movement?

• How does the player’s weapon work?
  – One bullet at a time? A spread? Upgrades?

• What will the collision shape of my player be?
  – What is fair?
Suggestions

• Allow the player to hold the fire button to continue firing

• Use the mouse to fire to avoid keyboard ghosting
TAC 3 Playtesting
More fun than last week!