Tic is over!
Tic Feedback

• Don’t fall behind!
Global Reqs

• They exist!
• Cover broad standards for every project
  – runs 20+ FPS, engine and game code separated, doesn't crash, etc.
• Will be publishing them to the website later today
Rubrics

- `/course/cs1971/rubrics/<project>`
- Copy over grade.txt for each checkpoint
- Makes grading easier for us
Playtesting

- We won't be having playtesting today
- Issues with our playtesting script
- Confusion over global requirements
- But no playtesting means you get to make up playtesting reqs!
Screen vs. UIElement

• Screens are NOT UIElements
• Similar methods, fundamentally different bits of software
• UIElements facilitate player interaction with the game
• Screens store everything about the game, not just things users can interact with
Handing In

• Please include project files and run the project once on a department machine
• If your eclipse project looks like this...
• ~/course
  – cs1971
    • tic
      – <login>
        » src
        » bin

Handin from here!
Reminder

• Have a generic Application.java in your engine
• Never import from your game package to your engine package!
This Week: Alc

• 2 week project
• Puzzle Game
• First checkpoint due next week
Game World

MOTIVATION
Games are busy...

- All games involve a number of game objects
  - May be many different types of objects
  - May be many instances of the same type of object
  - Maybe both of the above
- They exist in their own universe
  - If our entire universe is a game, you’re just an object
- We need to take the burden of representing and organizing these objects off the game code
High Level Representation

The GameObjects
- Small collections of functionality
- Hold their own logic and state
- Don’t know much about the rest of the game

The GameWorld
- The overarching collection of GameObjects
- Responsible for global logic and facilitating entity logic
- Represents the boundary between program and game
Without a Game World

ViewPort

GameScreen

Application

FrontEnd

Tile[][][]

Unit

Projectiles

AIUnit
With a Game World

ViewPort

GameScreen

Application

FrontEnd

World

Tile[][]

Unit

AIUnit

Bullet
RESPONSIBILITIES

Game World
What does a world do?

- Represents all GameObjects in a single space
- Centralizes object management
  - Maintains list of GameObjects
  - Passes ticks and draws to GameObjects
Multi-pass logic

- Ticking and drawing GameObjects in the wrong order leads to undesirable behavior
  - Drawing background over everything else
- World can selectively update state in order
  - E.g. tick all GameObjects so they update position, *then* check for collisions
  - Should have a way for specifying draw order
EventHandling

• Passing events can get cumbersome
• Combined mouse-key events can get complicated
• Solution- store input state in GameWorld
• Array of 256 booleans/enums instead of key events
• Keep track of mouse position/state
Game Worlds

QUESTIONS?
LECTURE 1
Engine Framework

*It’s possible that you might have a problem
Engine Framework

GAME OBJECTS
What is a game object?

- Everything your GameWorld holds
- Your background
- Your walls
- Your character
- Your enemies
They aren’t everything

- Your screens aren’t
- Your UI probably isn’t
- Saving/loading isn't
Hierarchical design

• Consider a simple game with:
  – Trees
  – Skeletons

GameObject
  • Has a sprite

Skeleton
  • Can move

Tree
  • Can grow
Hierarchical design

• Now we reach a problem
• Where does the Whomping Willow fit in?
Hierarchical design

• Can make a separate parent class and re-implement moving
• Can add below skeleton and add ability to grow
• Both not ideal
• Only get worse as the game gets bigger
Solution

• Component-based design
• Everything is a game object... and only a game object
• Leave game objects dumb
• Let their *components* do all the heavy lifting
Solution

- GameObjects are just lists of behaviors
- The behaviors implement all relevant functionality
- Composition over inheritance

**GameObject**
- List of behaviors

**HealthBehavior**
- Health bar

**MoveBehavior**
- Updates position

**KeyControlBehavior**
- Responds to key presses

**AIBehavior**
- Makes decisions

**DrawBehavior**
- Has a Sprite
Component-based design

- The appearance and logic of each object is defined by its components
- Making new objects is as easy as adding new components
GameObject Contract

• An object needs to:
  – Add a component
  – Remove a component
  – Access a component
  – Update each of its component
  – Tick ticable components
  – Draw drawable components
GameObject Contract

private ArrayList<
Component> _components;

public void addComponent(
Component c);

public void removeComponent(
Component c);

public Component getComponent(String tag);

public void tick(long t);

public void draw(GraphicsContext g);
Component Contract

• Needs to respond to ticks and draw events (can be empty)

```java
public void tick(long nanosSinceLastTick);
public void draw(GraphicsContext g);
```
Engine Framework

SYSTEMS
Systems

• Organize shared behavior
  – E.g., not all GameObjects will be drawn
  – GraphicsSystem should only hold GameObjects that have drawable behaviors

• Each System stores a list of relevant GameObjects and calls the relevant method on each of them

• E.g.
  – TimerSystem calls tick(long b) on GameObjects
Systems

• Your choice of when to register GameObjects with Systems
  – On instantiation of the relevant behavior
  – On addition to the GameWorld

• Your choice of how to register GameObjects with Systems
  – Manually, e.g. soundSystem.addObject(obj);
  – Automatically, e.g. have system ask each GameObject if it’s interested
Enforcing Draw Order

• Can set drawing layers
  – Array of arraylists
  – Add objects to relevant list based on draw order
  – onDraw, go through the array, drawing everything in the list at [0], then [1], etc.
  – Guarantees consistent draw order
Enforcing Draw Order

• Alternatively: use a TreeSet
  – log time lookup/insertion/removal
  – Predictable order (can set priority for drawing)
  – Adding the same GameObject twice won’t leave a duplicate
  – Requires custom comparator
Game Worlds

QUESTIONS?
Viewports

MOTIVATION
Sometimes screen space is hard

- Theoretically everything can be done in screen space
  - E.g. specifying size of objects in pixels, position in pixel offset
- But some things can be very hard
  - E.g. panning
  - Imagine four-player split screen
- Games shouldn’t have to worry about how the screen draws them
Game space vs. Screen space

- In nearly all games, it makes sense to think of the game as existing in its own “space”
- Add a UI element that has “view” into the GameWorld
- This is a viewport
Space Conversions

• Needs:
  – Upper left corner in screen space
  – Upper left corner in game space
  – Scale/zoom/pixels per game coordinate

• Game point to screen:
  1. Minus upper left in game space
  2. Multiply by scale
  3. Add viewport upper left

• Screen point to game:
  – Do the OPPOSITE of the steps IN REVERSE

Scale: 100 px/unit

<table>
<thead>
<tr>
<th>Screen</th>
<th>Game</th>
</tr>
</thead>
<tbody>
<tr>
<td>120, 20</td>
<td>1.5, 2.0</td>
</tr>
<tr>
<td>220, 140</td>
<td>1.0, 1.2</td>
</tr>
<tr>
<td>1.5, 2.0</td>
<td>120</td>
</tr>
<tr>
<td>2.0, 140</td>
<td>120</td>
</tr>
</tbody>
</table>
Implementing viewports

1. Store the clip and transform (g.save())
2. Set the clip
   a. You will draw outside the bounds of the viewport otherwise
3. Set the transform
4. Draw the game-space in its own coordinates
5. Restore the clip and transform (g.restore())
Setting the Clip

1. Set a path
2. Call `g.clip()`

- Easy way to set a path:
  - `g.rect(x, y, w, h)`
Setting the Clip

- Fancier Paths:
  - g.beginPath()
  - g.moveTo(x, y)
  - g.lineTo(x, y)
  - g.arcTo(x1, y1, x2, y2, r)
  - g.closePath()
More on Paths

• After setting up a path, you can also:
  – g.fill()
  – g.stroke()
Affine Transforms

- JavaFX’s **Affine** keeps track of geometric transforms for drawing
- Can create an **Affine** that converts from game space to screen space
  - Check out `appendTranslation()`, `appendScale()`, and others
- How to use the **Affine** once we have it
  - A **GraphicsContext** instance maintains an internal transform
  - The transform is applied to objects before they are drawn
  - Use `getTransform()` to get the current transformation to modify
  - Use `setTransform(...)` to set the modified **Affine**
Space Conversions

- **Affine** will handle these conversions!
- **Game point to screen:**
  1. Minus upper left in game space
  2. Multiply by scale
  3. Add viewport upper left
- **Clip guarantees you won't draw outside of the green box**

Scale: 100 px/unit
Affine Transforms

• If you ever want to do rotations in your game, you should use Affine – appendRotation()

• Never do any rotation calculations yourself
QUESTIONS?
Warnings!

• Viewports are essential to the rest of the class – every assignment from here on will depend on using your viewport!
  – Before you can playtest your game, you must have a functioning viewport
  – Design well
  – Test thoroughly
  – Don’t put off bugs until later weeks
• The TA staff requires the use of **Affine**
LECTURE 1
Tips for Alc I
Zooming

• Need some zoom factor
• Zooming is multiplicative, not additive
  – Rolling mouse wheel should * or / the scale factor
• Need to do more
  – Otherwise, zooming will focus on the top-left corner
Zooming on Viewport Center

- When converting game space to screen space:
  - Translate to viewport center in game space
  - Translate by \((-\text{viewportWidth} / 2, -\text{viewportHeight} / 2)\)
  - Scale by zoom value
  - Translate by \((\text{viewportWidth} / 2, \text{viewportHeight} / 2)\)
Zooming on Mouse Center

• Make a new Affine
• Calculate new scale based on zoom
• Recalculate upper left corner in game coordinates
  – newX = gameMouseX - (screenMouseX - oldX)/scale
  – newY = gameMouseY - (screenMouseY - oldY)/scale
• Apply transformations to Affine like when you first created it
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` 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 Jeff Hao has an excellent chapter on generics
- Gives examples of where advanced generics are useful
- Can be found in the back of the SunLab
Factories

- Static functions that initialize objects
- Useful for creating standard templates for game objects without subclassing

```java
public static GameObject makeElementFire() {
    GameObject o = new GameObject();
    o.addComponent(new ElementComponent("fire"));
    ...
    return o;
}
```
Tips for Alc I

QUESTIONS?
TIC PLAYTESTING

(next week)

Yay!