CSCI 1106
Lecture 21

Game Design Review
Components of a Game

- **Stage:** Displays (renders) the game
- **Sprites:**
  - Graphical objects that interact on the stage
  - Represent various artifacts in the game
    - Characters
    - Projectiles
    - Power-ups, obstacles, etc
- **Game Code:**
  - Governs interactions between sprites
  - Governs interactions between player and sprites
  - Implements the rules of the game
  - Contains *event handlers* that respond to events in the game
  - Updates the sprites on the stage
The Movie Metaphor

• In a movie the screen is updated 24 times per second
• In a game the stage is updated 30 times per second
• The update is called a *frame*
• A frame occurs every $\frac{1}{30}$th of a second
• When a frame occurs
  – Sprites modify their properties
    • Position
    • Look
    • Sound
    • Etc
  – Sprites are redrawn on stage in each frame
• Key Idea: A game is simply an interactive movie!
• What interaction?
Event Driven Paradigm

• Observation: A game performs “some action” when “something” happens

• Idea: Game code simply responds to events

• Possible events:
  – External events: (mouse, keyboard, kinect, etc)
  – Internal events: (Start of game, New Frame, Timer)

• Each event is handled by an event handler

• The game code simply consists of event handlers that handle all aspects (behaviours) of the game!
The Main Loop

- **Idea:** The main loop is implemented for you

- **Main Loop:**
  - Event (action) occurs
  - Handle (respond to) event
  - Update (modify) object(s)

- **All you need to do is**
  - generate events and
  - write the event handlers!
Sprites

• A sprite is a graphical object that is placed on the stage
• A sprite has associated with it
  – *costumes*
  – *properties*
  – *variables*
  – *scripts*
• A sprite represents game artifacts
  – Characters
  – Obstacles
  – Projectiles
  – Etc
Properties and Variables

Sprite Name: Invader

Costume1

Properties:
- 10: x position
- 42: y position
- 90: direction
- 100: size

Variables (Extrinisic Properties):
- Score: 123
- Level: 4
- Speed: 5
- Lives: 2

Costume2
The Stage

- Idea: The *Stage* is a special sprite on which all other sprites are displayed.
- The stage has *backdrops* rather than costumes, but they serve the same purpose.
- All sprites will always be in front of the stage.
- Like other sprites, the stage has
  - properties, sounds, and scripts associated with it.
Cloning Sprites

- Idea: We can make multiple copies of a sprite by cloning it.
- When a sprite is cloned, everything is copied e.g., properties, variables, costumes, scripts, etc.
- Key Idea: Manipulation of the clone or the original does not affect the other e.g., changing the clone's position will not move the original.
- Both the clone and the original have the same name.
- Two differences between clones and originals
  - clones are notified when they are created
  - clones can be destroyed
Communication Between Sprites

• **Key Idea:** Sprites communicate by broadcasting messages (events)

  – A broadcast means *every* sprite receives the message

    e.g., Stage broadcasts FRAME 30 times per second

  – A sprite can respond to a specific message (event) by having a script that receives it

• **Messages cannot be directed at a specific sprite unless only that sprite has a script to receive that message**
Autonomous Motion

- Set the sprite’s speed
  - Number of steps (pixels) per frame
  - *Can be positive or negative*

- Set the sprite's direction property

- Create a script to respond to the FRAME event

- On each frame change the position of the sprite by its speed

  e.g. move 10 steps per frame at 90°
Hitting the Wall

• Fact: If the object keeps moving it will reach the edge of the stage
  – Fall off the edge
  – Bounce back

• Falling off the edge
  – Once object is no longer visible, remove it

• Bounce back
  – Once object touches a wall, reverse velocity
    • If vertical wall, reverse horizontal velocity
    • If horizontal wall, reverse vertical velocity

• This is done in the FRAME handler
  – Why?

• This is a special form of collision detection
Mechanisms for Collision Detection

• Four ways to detect collisions:
  – Cheap and fast: Check if bounding boxes overlap
  – Expensive and slow: Check if the points of one sprite intersect with the other
  – Fast but specialized: Use geometry
  – More complicated and fast: Use invisible sprites

• For most purposes, the second way suffices
**Player Motion**

• All interactive games have player movement
  – Players can move their character or avatar on the screen
  – Players can react to the game and move their avatar

• How the avatar moves is dictated by the game’s
  – Laws and physics of the game
  – Goals and objectives
  – Environment and level of play

• Common ways to move the avatar are through
  – Mouse
  – Keyboard
  – Dedicated game controllers and joysticks
Mouse Movement

**Direct Mouse Movement**
- The avatar appears where the mouse is pointing to
- No need to control the velocity of the avatar
- Position and velocity is managed by the mouse movement
- Set the avatar’s coordinates to the mouse coordinates at each FRAME event

**Easing**
- Gradually move avatar toward the location clicked on with the mouse pointer
- A mouse click sets the target to move toward
- Calculate distance between the avatar and target
- Incrementally move the avatar toward the target

(set x to mouse x)
(set y to mouse y)
Keyboard based Movement

- Idea: Move the player with the keyboard
  - The arrow keys control the direction that the avatar moves
  - These directions allow the player to move diagonally as well
  - Need to respond to the KEY PRESS events or check if keys are being pressed.
  - More than one key can be down at the same time

- On a FRAME event
  - Check which of the arrow keys are pressed and move in that direction
Playtesting

• **Playtesting is a game development method for**
  – Getting feedback about the game
  – Identifying problems with the game
  – Understanding how players perceive the game
  – Improving the playability and enjoyment of the game

• **Playtesting involves**
  – **Players:**
    • Users who typically have never played the game before
    • Recruited by developers to play games
  – **Observers:**
    • Members of the development team
    • Observe the players as they play games and take notes
Goals of Playtesting

• Identify game play issues
  – Bugs
  – Playability: Player motion and mechanics, Environment, Controls, Speed of the game
  – Understandability: Game objectives, Tactics and strategies, Player information and statistics

• Understand how players perceive the game
  – Difficulty
  – Pace
  – Immersion
  – Interest (story line)
  – Genre

• Get feedback about the game

• Identify possible improvements
  – Extensions
  – Modifications
  – Spin-offs
  – Features
Playtesting Process

**Things to do**

- **Before the playtest**
  - Ensure the game is stable
  - Recruit players
  - Setup a “typical” game station
- **During the playtest**
  - Welcome and **thank** the player
  - Remind the player that they are not being tested
  - Ask the player to talk as they play
  - Remain silent and take notes
  - Thank the player again ensure that you have contact information
- **After the playtest**
  - Keep track of all the players
  - Categorize your observations

**Things to note**

- General mood of the player
- Any comments or suggestions made by the player
- Any bugs that occur during play
- Any struggles experienced by the player
- How easily the player learns the game
- How quickly does the player progress through the game
- How quickly does the game become too hard for the player
- Any aesthetical issues
- Any other feedback
High-Level Game Design

• Game Elements
  – Mechanics
  – Story
  – Technology and Aesthetics

• Idea: The elements work together to create a unifying theme in the game
  – What experience do you want to convey?
  – Structure your story and mechanics to reinforce the theme
The Game Story

• There’s nothing like a good story to pull you in...
• A story is composed of:
  – A "world"
  – Characters
  – A quest
• The story immerses the player and separates great games from ok games
• Story Considerations
  – Depth: How detailed or grand is the story to be?
  – Delivery: How is the story communicated to the player?
  – Pacing: How quickly is the story being told?
Game Mechanics

• Idea: Use game mechanics to
  – Implement the game story
  – Support the unifying theme of the game

• Game mechanics comprise
  – Rules: Written/Unwritten/Game objective
  – Environment: Space/Number of players/Physics
  – Actions: Primitive vs Strategic
  – Chance (Randomness): “Secret of fun”
  – Skills: Physical/Mental/Social

• Idea: A set of stock (standard) mechanics that are used by similar games is called genre
  – Card games, Racing games, First-person shoot-em up

• Idea: Use state transition diagrams to model game mechanics
Game Mechanics: Rules

• Written rules of play (what happens when I...)
  – User manual
  – Game code
• Unwritten rules
  – Etiquette
  – Sportsmanship
• Object of the game (how do I win the game)
  – Clear
  – Achievable
  – Rewarding/Fun
Game Mechanics: Environment

• **Spaces**
  - Discrete or continuous?
  - Boundaries?
  - Nested Spaces?

• **Number of players**
  - Computer
  - Human

• **Physics**
  - Interaction of objects
Game Mechanics: Actions

• Primitive Actions (private’s view)
  – Moving the player
  – Shooting

• Strategic Actions (general’s view)
  – Protecting a zone
  – Ambushing

• Most games require combination of both types of actions
Game Mechanics: Chance

- Adds a surprising or unexpected elements
  - The so called "secret of fun"
- Consider how probabilities will factor into the play over the duration of the game
  - Power-ups
  - Density of projectiles
- Some predictability is useful! Why?
- The “chance trade-off”
  - A lot of randomness: game is about tactics, short term
  - A little randomness: game is about strategy, long term
  - Good games have the right mix
Game Mechanics: Skills

• Idea: The right amount of challenge will keep the player interested

• Three types of skills:
  – Physical Skills
    • Strength, dexterity, coordination, and endurance
    • E.g. How fast can I hit that button?
  – Mental Skills
    • Memory, observation, and problem solving
    • E.g., The answer is ...
  – Social Skills
    • Reading and fooling opponents
    • Coordinating with teammates

• Many successful games combine skills from multiple categories
Project Management

The Problem
• A project consists of many parts
  – Tasks
  – Goals and milestones
  – Dependencies
  – Resources
  – Risks
• To complete a project
  – Finish all tasks on time
  – Accomplish all goals
  – Satisfy all dependencies
  – Use only the allocated resources
  – Adapt to things going wrong

The Solution
• Things to consider
  – Tasks take a set amount of time
  – Some task must precede other tasks
  – Resources are limited
  – Things go wrong

• Things to do
  – Identify and schedule tasks
  – Allocate resources
  – Anticipate and manage risks
  – Complete a project on time and on budget
Identifying the Tasks

• A task
  – Takes a minimum amount of time to complete
  – Requires specific resources
  – Requires certain other tasks to be completed first
  – Must be completed before other tasks can begin
  – May take longer than expected due to unanticipated events

• For each task identify
  – What the task is
  – What resources it requires
  – What tasks does it depend on
  – How much time it will take

• Idea: Work backwards (reverse engineering)
  – Start with the end goal
  – Ask what task(s) are needed to achieve the goal
  – Ask what resources are needed for the tasks
  – For each task break it down into subtasks and repeat
Scheduling Tasks

• Problem
  – There are many tasks
  – There are many resources
  – Each task may have multiple dependencies

• Need to
  – Organize all tasks in one place
  – Sort dependencies
  – Check for resource contention

• Idea: Use a Gantt chart
**The Gantt Chart**

**The Purpose**
- Represent all tasks
- Represent resource use
- Represent dependencies
- Represent time of tasks

**Gantt Chart Rules**
- Time is represented horizontally (left to right)
- Resources are denoted vertically
- A task requires both time and resources
  - Represented by one or more rectangles
- If two tasks require the same resource, they cannot overlap
- If task A depends on task B, task A must follow task B
- The minimum amount of time needed to fit in all the tasks is the minimum amount needed for the project
Scheduling Issues

• Dependency chains
  – Task A depends on B depends on C depends on D ...
  – Time of longest chain is the minimum time of the project
  – Place longest chain first
  – Then the next longest ...

• Resource contention
  – Tasks cannot use a resource at the same time
  – A bottleneck occurs when many tasks need the same resource
  – Stagger tasks to avoid resource contention
  – Add more resources to reduce contention

• Risk Management
  – Schedule tasks as early as possible to provide time to deal with unforeseen events
  – Schedule extra time for each task
Using Randomness

• Idea: Most systems have a pseudorandom source of values
  – The source is an infinite sequence of values
  – The values look random
  – Are sufficiently random for our purposes

• Each system is a little different, but all work similarly
  – Each system provides a Random function
  – The function returns a value chosen randomly from a fixed range
in Scratch

• Scratch has a function
• Returns a value in the range \( \min \leq n \leq \max \)
• Value is selected at random from a uniform distribution
• What does a uniform distribution mean?
Projectiles

- **A projectile**
  - Appears on the stage when the player/opponent does something
  - Appears initially at the player/opponent’s location
  - Moves away from the player/opponent in a set direction
  - Disappears when it hits something
  - Causes opponent/player to react in some way

- **Projectile Life-Cycle**
  - Initiation: Determine when the projectile is to be created
  - Creation: Create, position, and launch the projectile
  - Motion: Move the projectile along the stage
  - Collision: Check if collisions occur and respond to them
  - Elimination: Remove projectile if it collides or leaves the stage
Projectile Initiation and Creation

Initiation

• Idea: A projectile is initiated as a result of an event
  • Player events:
    – Mouse click or key press
    – Collision with another object
  • Game (opponent) events:
    – Random or regular time intervals
    – Collision of objects within the game
    – Start of game or level (e.g., the ball in BrickBreaker)
• Idea:
  – Broadcast NEW_PROJECTILE when a projectile is needed
  – The projectile sprite will receive the event and create the projectile

Creation

• Idea: Projectiles are created by an event listener
• To create a projectile
  – Projectile sprite
    • Receives NEW_PROJECTILE
    • If sprite is not a clone and a projectile can be created
      – Set position
      – Set speed
      – Set direction
      – Clone self
  – Projectile clone
    • Marks itself as a clone
    • Set itself as visible
Projectile Motion and Collision

Motion

- **Idea:** Projectiles move just like all other objects
  - Add velocity to position on each FRAME event
- **Idea:** FRAME handler may also
  - Adjust velocity of projectile as game mechanics dictate
- **Note:** The original projectile sprite should never move and always remain hidden

Collision

- **Idea:** Purpose of projectiles is to collide!
- **Idea:** On FRAME events
  - If projectile has collided with a game object
    - Create some special effects
    - Adjust state of game object
    - Remove projectile from stage
  - if projectile has moved off-stage
    - Remove projectile from stage
Projectile Elimination

• Idea: Once a projectile moves off-stage or has collided, remove it!
• Your game will slow down if you do not!
Buttons

• Buttons are screen objects that identify an action and how to perform it
• Buttons identify an area for a user to click on
• Buttons generate an event that the application can respond to by running a listener
• A button has three (3) states
  – **Up** is the normal state of the button
  – **Over** is when the mouse is hovering on the button
  – **Down** is when the button is pressed
• Idea: For each of the three states the button can have a different look
• Idea: When the button changes state, it generates an event
Creating Buttons

- Create *sprite with three costumes*
  - *Up*
  - *Over*
  - *Down*
- Have sprite receive FRAME event
  - If the mouse is touching the button
    - If clicked *[Down]* use Costume 3
    - Otherwise *[Over]* use Costume 2
  - Otherwise *[Up]* use Costume 1
- Only change costumes if necessary!
- When should we actually execute action associated with button?

*when this sprite clicked*
It is useful for games to display text

- Instructions
- Player information (score, health, level, etc)
- Dialogue

There are two types of text that we can display

- Static text, which does not change during the game
  - Instructions
  - Dialogue

- Dynamic text, which changes as the game progresses
  - Player information
Game Polish: Motivations

• A polished game is
  – More compelling and immersing
  – More likely to be played longer
  – More appealing to new players

• A polished game will
  – Get better reviews
  – Get more praise on social media and word of mouth
  – More likely become popular
  – Likely sell more copies

• It’s in our interest to make sure that games are as polished as possible!
Game Polish

• Defn: A process to reduce the number of minor issues associated with the game
• This involves
  – Fixing minor bugs and anything that detracts from the consistency of the game
  – Touching-up graphics
  – Refining game mechanics
  – Adding minor features and special effects
• Idea: Schedule game polishing as part of your overall development plan
• Should be done throughout the game development cycle
• Done in concert with playtesting
Types of Game Polish

• Resolution of issues (1\textsuperscript{st} Priority)
  – Stability
  – Consistency
  – Playability
  – Understandability

• Refinement of the game mechanics (2\textsuperscript{nd} Priority)
  – Realism, environment, and actions
  – Graphics
  – Audio

• Additional features (3\textsuperscript{rd} Priority)
  – Special effects
  – Side stories and bonus rounds
  – Easter eggs
  – Special objects