Software Studio

軟體設計與實驗

Game Design Techniques

Hung-Kuo Chu

Department of Computer Science

National Tsing Hua University

CS2410

Game Design

- Game Design is a skill that requires a lot of experience in **playing games**, **seeing how others play games**, and **making games**.
- We can't teach you game design, but we can introduce to you some **game design techniques** often used in **good 2D games**, and you can go and research the ones you are interested in.

How to use this document?

- Give a quick look at every technique listed.
- Read the attached memo under every page.
- If there’s a technique that piques your interest, feel free to go and research further. Look for other examples and try implementing the techniques yourself!

Physics

- Built-in (dynamic) physics VS custom (kinematic) physics.
- **Dynamic physics:**
  - Slightly easier to set up.
  - "Realistic" physics is a double-edged sword.
    - Physics-based features such as slippery floor and knockback.
    - Simple tasks like climbing a slope are constrained by friction.
  - Hard to control precisely.
    - Not recommended for platformers if you want **good controls**.
- **Kinematic physics:**
  - Hard to set up.
    - You have to implement gravity and collision resolution.
  - Easier to control precisely.

Physics

- **Mixed approach: Kinematic physics** when moving, **dynamic physics** otherwise.
- Achieved by **always directly assigning the player's velocity when moving**, rather than applying forces or impulses to affect it indirectly.
  - Effectively no friction when moving.
- Experiment to see which kind of physics your game would work best with!

Coyote Time

celeste can i go to edge?
to fall off edge?
CELESTE
yesss
actually jumps off air like a boss
coyote time

Source: r/celestememes

Coyote Time

- A technique used to give players **leniency** in terms of jump timing.
- Lets the player still jump if they just **left a platform recently**.
- See **here** for a video explanation.
- **Letting the player jump twice or more** can also achieve the same effect, to a lesser extent.

Vector Fields

- Areas where **force** is applied.
- Two types:
  - **Uniform:** Constant force everywhere in the field. Usually used to create simple **wind** or **buoyancy**.
  - **Non-Uniform:** Varying force depending on the position in the field.
    - **Gravitational/Magnetic force:** Inversely proportional to the distance to a "source."
      - Example: Angry Birds Space

Camera

- **A good camera shows the player what the developers want them to see.**
- Common camera techniques:
  - Camera movement:
    - Snap-to-position / interpolate
    - Offsets
    - Follow target
      - Soft zones, dead zones
    - Fixed-path (camera dollies)
    - Bounds
      - Camera "collision"
    - Shake
  - Camera zoom in/out
- Check out Unity's Cinemachine demo to see the effects in action!

See also: GDC 2015 – How Cameras in Side-Scrollers Work

Perspectives

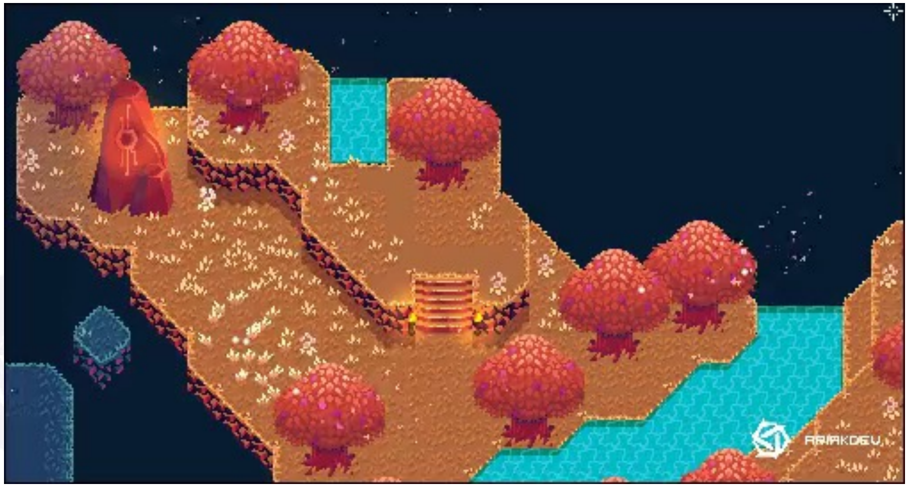
- How the camera is "positioned" relative to the game world.
- In 2D games, mainly **top-down** and **side-view**.
- **Top-down:**
  - Usually involves little to no physics.
  - Often associated with puzzles and dungeon crawlers.
- **Side-view:**
  - Usually involves physics.
  - Often associated with platformers.

2.5D Perspective

- In a 2D engine, this means creating **fake depth** in the scene.
  - Objects interact (mostly) on a fixed plane (XY or XZ).
  - Use techniques such as **Z-sorting** to add the illusion of a third axis to the scene.

Top-Down 2.5D Perspective

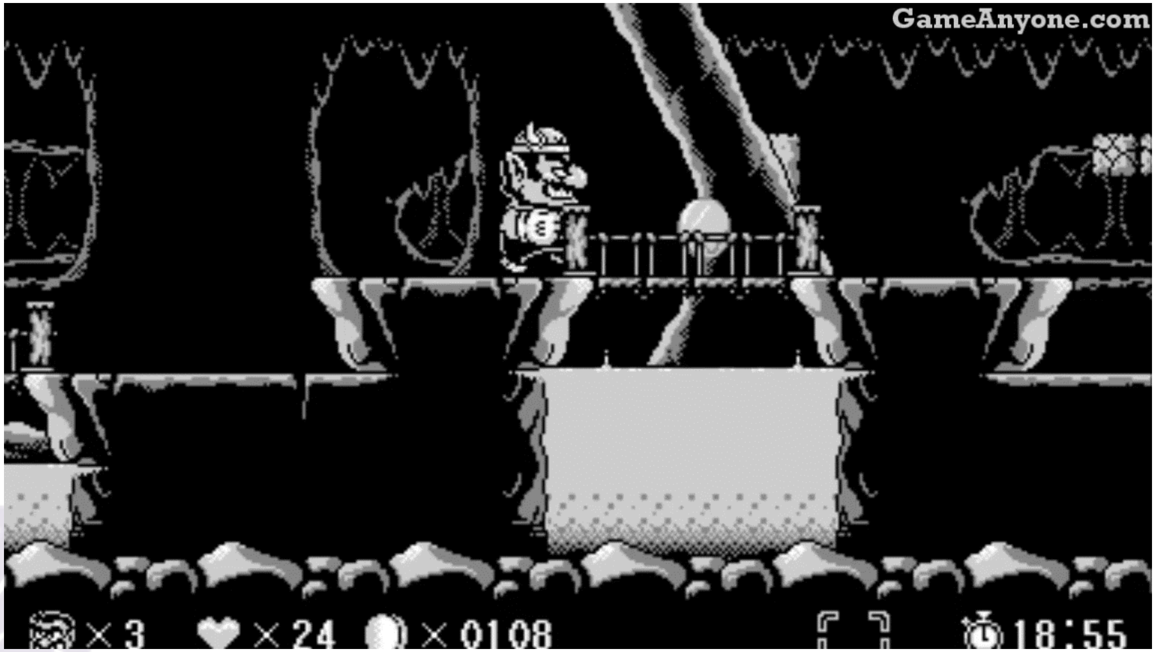
- A combination of 2.5D techniques on a top-down tilemap:
  - **Wall tiles** are used to make the hill in the middle look like it’s taller than the ground below it.
  - The player’s Y-coordinate **combines both their coordinate on the tilemap as well as their “height” on the fake Y-axis**.
  - The player’s Y-coordinate is offset downwards slightly in shallow water tiles to make the player “submerge” into the water.
  - The tree is rendered **after** the player when they go behind it.



Source: YoYo Games

Side-View 2.5D Perspective

- Use **size** to tell the player that an object is far away.
- Two tilesets for different “distances” to the camera.



Source: Virtual Boy Wario Land

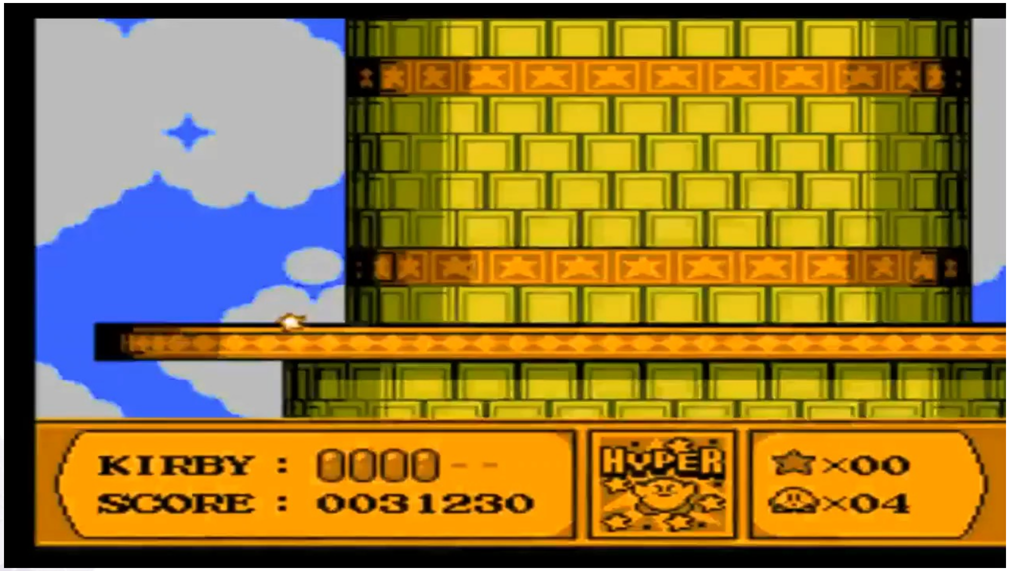
Parallax Scrolling

- When moving in one direction, objects **closer** to you appear to move **faster**.
- Typically done by having multiple scrolling background layers at different speeds.
- Make far layers lighter in color to create **fog**.
- Can be **horizontal** or **vertical**, or **both**.

Source: Wikimedia

Cylindrical Scrolling

- **Animated tiles or backgrounds** can create the **illusion** of objects **moving around a cylinder**.



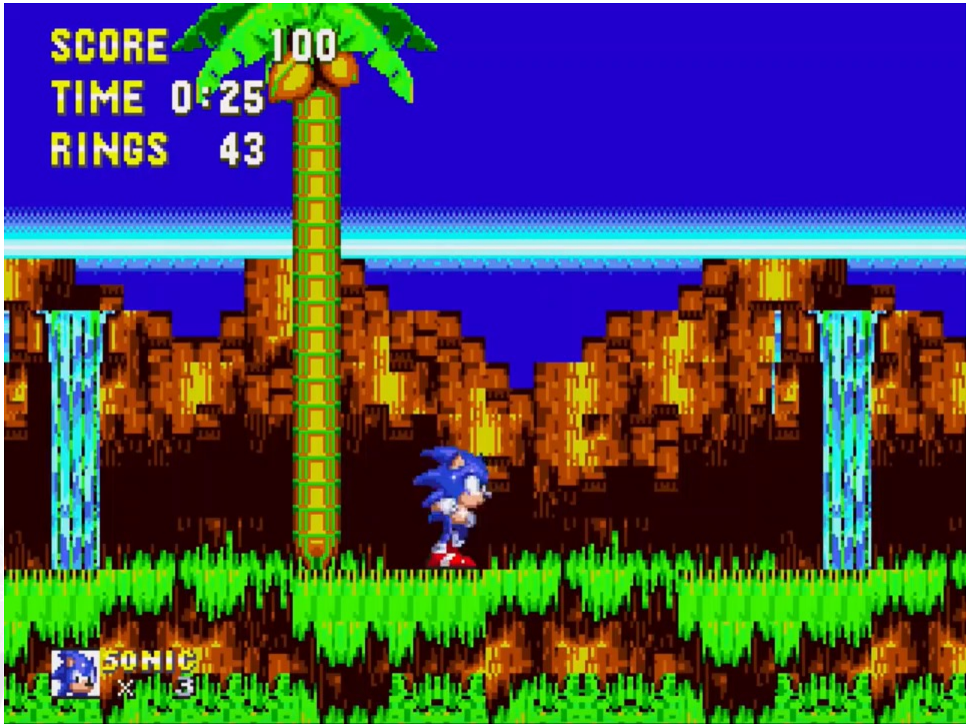
Source: Kirby's Adventure
The tiles used to create this effect.

Transitions

- Visual effects that make changing scenes or level sections look smooth.
- Check out some transitions in Powerpoint or Google Slides and try replicating them by **moving images of black boxes** around.
  - **Sometimes a simple “fade in/out” might not be the best option!**

Transitions

- **Particle effects** that quickly block the camera can also be an effective transition!



Source: Sonic the Hedgehog 3 (0:45~)

Lighting

- Realistic 2D lighting requires **ray casting** or **shadow mapping**, rendering techniques that are out of the scope of this course.
- Old hardware didn’t have the power to perform the complex computations needed for realistic lighting, so game developers used other simpler techniques to create **fake lighting** instead.
- We can take a few lessons from them!

Fake Lighting

- By overlaying the scene with a black image, we can create a fake spotlight.
- Treat the alpha channel (transparency) of a pixel as "brightness."

Source: Super Mario World 2 - Yoshi's Island

Fake Lighting

- Different shapes of the mask can create different lighting effects.

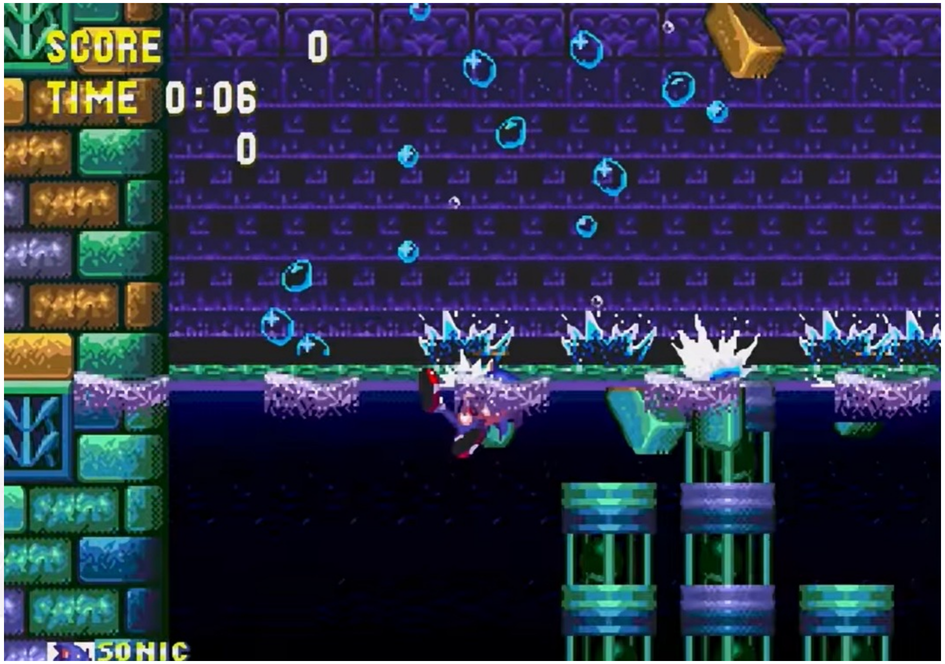
Source: Super Mario World

Water

- Realistic 2D water requires 3D **mesh manipulation** techniques, which are out of the scope of this course.
- Much like lighting, game developers used to use overlays to give underwater objects a different hue.

Fake Water

- Underwater objects have an aqua-colored hue.
  - Replicate this effect by experimenting with **blend modes** (part of import settings).
- Water ripple **animations** are spawned when an object crosses the water surface.
  - You can try using **particle effects** here too!



Source: Sonic the Hedgehog 3

Artificial Intelligence (AI)

- Programs or scripts that try to achieve a certain goal over time.
  - It could be to **hurt** the player (by touching them, shooting them, etc.)
  - It could be to **help** the player (by retrieving useful items, attacking enemies, etc.)
  - It could be to give the player a **challenge** (by giving the player a fair fight)
- It is important to remember that in most games, AI is used to give the player a **good experience** rather than beating them all the time.

Artificial Intelligence (AI)

- Three main algorithms:
  - **Rule-based:** A table of **if-else statements**, telling the AI what to do in each situation.
    - Simple, but susceptible to edge cases. Scales poorly for complex problems.
  - **Search algorithm:** Given the current game state, the AI takes the "best" move according to a set of **heuristics**.
    - Used to be the most common AI algorithm before modern hardware enabled real-time use of ML.
  - **Machine learning (ML):** This is out of the scope of our course, so we won't discuss it here.

Rule-based AI

- Often used for **simple enemy behavior**.
  - “Walk forward. Turn around **if about to fall off a platform**.”
  - “**If player is in line of sight**, fire a bullet at them.”
- Older RPGs such as *Final Fantasy* combined rules with **probability** to be less predictable.
  - “**If HP > 50%,** 70% chance to use a normal attack, 20% chance to use a magic attack, 10% chance to heal self.”
  - “**Otherwise**, 40% chance to use a normal attack, 40% chance to use a magic attack, 20% chance to heal self.”

Search algorithm

- Often used for **pathfinding**, and computer opponents for **puzzle games** and **strategy games**.
- The exact type of algorithm is selected based on the type of game.
  - For games with relatively **shallow** state space (mainly turn-based games, where decisions are made every turn), you can try stochastic algorithms such as the **Monte Carlo Tree Search**.
  - For games with **deep** state space (mainly real-time games, where decisions are made **every frame** or **every few seconds**), you can try the **A* algorithm** with a heuristic algorithm based on your understanding of your own game.
- Introduce **noise (random numbers)** to a search algorithm to weaken the AI.

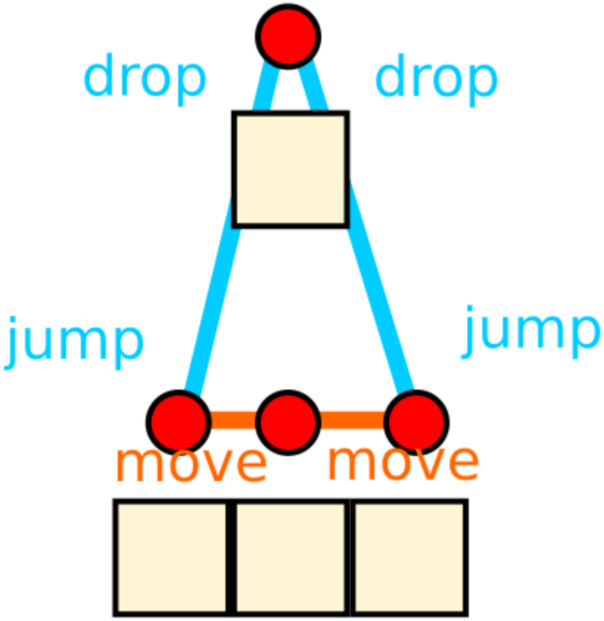
Pathfinding

- A special AI problem where the objective is to find the shortest path to a given location.
- For example, in a horror game, there might be a ghost or monster, **chasing the player**.
- **Constraints** in the environment and the **actions** the AI is allowed to take affect how hard it is to implement a pathfinding algorithm.

Pathfinding

- For a grid-based game, the Manhattan distance is a good heuristic to be used with the A* algorithm.
- Otherwise, you will need to transform your scene into a **navigation mesh (navmesh)**.
  - A graph where the vertices are **locations** in the scene, and the edges are the **actions** the AI needs to take to go from one vertex to another.
- See **here** for a real example.

Pathfinding



A graph that shows an AI how to navigate around a small scene.
Source: Game Development Stack Exchange

Procedural Generation

- A powerful technique to make your game **endless**.
- A lot of popular mobile games use procedural generation to create **“endless runner”** games.
  - Examples: Subway Surfer, Jetpack Joyride, Temple Run, Flappy Bird
- Also used to generate **dungeons** and **terrain**.

Endless Runner

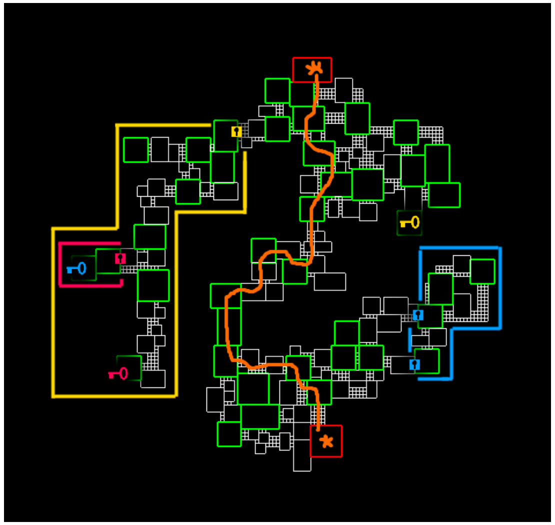
- The simplest form of procedural generation. **Linear**.
- From a table of patterns (possibly **prefabs**), randomly pick one to be the next pattern sent in the player’s way.
  - These patterns could be generated during runtime (like the pipes in Flappy Bird).
- Introduce more randomness by randomly offsetting some objects in an instantiated pattern.

Dungeon Generation

- The backbone of the popular game genre **“Roguelike”**.
- Decompose a dungeon into rectangular **rooms**.
- Prepare a **preset** of rooms, and for each room, **define the rooms they can connect with**.
  - This definition can be explicitly set, or implicitly determined by inspecting the tiles for free space.
- Under these connectivity constraints, randomly select rooms to add to the dungeon.

Dungeon Generation

- Even the rooms themselves can be procedurally designed using **Binary Space Partitioning**.
- Combine with preset patterns to create more interesting variations!



A procedurally-generated dungeon with keys and locked doors
Source: Phigames

Terrain Generation

- We can use the **midpoint displacement algorithm** to create realistic terrain.
- Start with two connected line segments, repeatedly add midpoints to each segment and randomly displace the midpoints' heights with a random number.
- You can then convert the contour into tiles or colliders.

Terrain Generation

A procedurally-generated terrain using the midpoint displacement algorithm.
Source: Douglas Paul

Multiplayer

- Letting multiple **human** players play with one another.
- Two types:
  - **Local:** The players must use the same machine physically. (ie. They're in the same room physically)
    - Game developers used to be forced to do so because the Internet was slow back then.
  - **Online:** The players can use the Internet to play with each other.

Local Multiplayer

- Simple to set up.
- For the keyboard, a typical approach is to create two sets of controls for your game, one being **WASD-based (left side),** the other being **Arrow key-based (right side)**.
- If the pandemic gets worse again, this is probably not going to be an option. ☹

Online Multiplayer

- Hard to set up, but a lot of teams in previous years have succeeded.
- Two options:
  - Use **Firebase** to support serverless multiplayer. (Like your midterm assignment)
  - Use an existing API such as **Photon**.
    - Can scale to support many players.

User-Generated Content

- A feature that lets players create content for the game easily.
- A game that actively supports UGC can create a community around it quickly.
- A classic and popular feature that encourages UGC is the **Level Editor**.

Level Editors

- In-game systems that lets the player use a set of **tools** and **objects** to **build** custom levels and **share** them with others.
- Examples: *Minecraft*, *Terraria*, *LittleBigPlanet*, *Super Mario Maker*
- Combine with cloud storage to let users share their levels even more quickly.

thank you!