GAME DEV ROADMAP

General Objective: Enable beginner developers to progress from basic to advanced levels through practical, progressively challenging projects. The focus is exclusively on 2D games, without imposing specific engines — you may use Godot, Unity 2D, Pygame, or any other of your preference.

Program Structure:

- **Core:** 20 mandatory, identical projects aimed at consolidating essential fundamentals.
- **By Genre:** 5 projects per genre, with 2 mandatory and 3 optional; inspired by real games, featuring minimum deliverables with opportunities for extension.
- **Game Jams:** Interspersed sessions for intensive practice, creativity, and rapid experimentation.
- **Game Modes:** Distributed among solo, co-op (local or online), and versus (local or online), depending on the genre.
- **Difficulty Scale (1–5):** Assesses the implementation complexity for a developer working alone.

CORE — 20 Mandatory Projects (Exact Copies)

- **Objective:** Master the essential fundamentals of 2D game development.
- **Content:** Input handling, loops, collisions, scoring, game over screens, and other basic mechanics.
- **Methodology:** Completion of 20 identical projects to ensure consistent practice.
- **Difficulty:** Progressive, ranging from 1 to 5 on a 10-point scale.

1 Copy of Pong

General Description

This project consists of reproducing the classic game Pong, where two paddles controlled by players or AI bounce a ball back and forth within a confined screen. The ball moves at a constant speed and changes direction upon collision with the paddles or side walls. Each player has a score that increases when the opponent fails to hit the ball. The main goal of the project is to learn and practice the fundamentals of 2D games, including movement, collision, boundary detection, scoring logic, and round resetting.

This is an ideal project for beginners who want to consolidate basic game programming skills, as it combines simple physics, player input control, basic AI, and real-time visual interaction.

Genre

Arcade / Pong-like

Typical features: fast-paced matches, simple mechanics, immediate visual feedback, and emphasis on player reflexes and precision.

Difficulty

1/10 for solo implementation.

The complexity is low but allows for additional layers of difficulty as the developer gains confidence, such as advanced AI, spin effects, and additional game modes.

Game Modes

- **Solo vs. AI:** The player competes against the computer.
- **Local Versus:** Two players share the same screen, each controlling a paddle.

Possible Extensions (Optional):

- Online cooperative or competitive: networked matches with remote players.
- Local 2v2: two paddles per side, increasing gameplay complexity.

Minimum Deliverables

To consider the project complete, all of the following elements must be implemented and functioning correctly:

Player Paddles

- Responsive vertical movement, with optional acceleration for smoother control.
- Screen boundaries respected, preventing paddles from leaving the play area.

Ball

- Constant linear movement.
- Collision with top and bottom walls: Y-axis inversion.
- Collision with paddles: X-axis inversion, potentially varying the bounce angle depending on impact position.
- Automatic reset to the center after a point is scored.

Score

- Visible score display on the screen.
- Match played up to a defined limit (e.g., 11 points).
- Score updates after each point.

Rounds and Reset

- Automatic ball reset after a point.
- Preparation for the next round without freezing or glitches.

Optional Extensions

These features are optional but enrich the practice and increase project complexity:

- Adjustable AI: varying difficulty of the computer-controlled paddle.
- Ball spin: modify bounce angle based on paddle movement at impact.
- Local 2v2: two paddles per side, requiring coordination and strategy.
- Replays or visual effects: enhance game aesthetics and allow match analysis.
- Online modes: learning about networking, state synchronization, and latency.

Learning Objectives

Upon completing this project, the developer should be able to:

Player Input

- Read keyboard or joystick inputs and move objects on the screen.

Basic 2D Physics

- Implement simple collisions and realistically alter object directions.

Scoring and Round Logic

- Manage game states, scoring, and automatic round resets.

Simple AI

- Create automatic behaviors for the opponent paddle (e.g., follow the ball with limitations).

Screen and Graphic Element Management

- Draw objects, update their positions each frame, and organize visual layers.

Optional Extensions (if implemented)

- Understand networking techniques, visual effects, and enhanced 2D physics with spin or dynamic acceleration.

This project serves as a solid foundation for any beginner in 2D game development. Mastering an exact copy of Pong establishes the groundwork for more complex projects such as platformers, shooters, or real-time strategy games. The key is to internalize movement, collision, scoring, and visual feedback—concepts that are universal in almost all digital games.

2 Exact Copy of Breakout

General Description

This project consists of recreating the classic Breakout, an arcade game where the player controls a paddle at the bottom of the screen, moving it horizontally to bounce a ball that destroys blocks positioned at the top of the screen. Each block can have one or more hit points, meaning the number of impacts it can withstand before being destroyed. The game may include simple power-ups, such as increased paddle width or extra balls, which appear randomly after destroying blocks.

The main goal of the project is to learn how to handle collisions between multiple objects, implement interaction logic between game elements, and manage levels, while consolidating fundamental 2D game concepts such as basic physics, player input, and scoring or progression systems.

Genre

Arcade / Brick-breaker

Typical features: fast-paced matches, reflex- and precision-based challenges, progressive difficulty across levels, and rewards (power-ups).

Difficulty

1/10 for solo implementation.

The project is relatively simple but offers opportunities for gradual enhancement, such as multiple block types, varied power-ups, and a level editor.

Game Modes

- **Solo:** The player controls the paddle alone and attempts to complete all levels.

Optional Extensions:

- Local Coop: two players controlling separate paddles on a split or shared screen.
- Challenge or Infinite Modes: score as high as possible until all lives are lost.

Minimum Deliverables

To consider the project complete, all of the following elements must function correctly:

Player Paddle

- Responsive horizontal movement.
- Screen boundaries respected to prevent leaving the play area.

Ball

- Constant movement and accurate bouncing off walls and the paddle.
- Collision with blocks: ball changes direction upon impact.
- Reset upon losing a life (if a life system is implemented).

Blocks

- At least 3 complete levels of blocks.
- Each block may require 1 or 2 hits to be destroyed.
- Visual update of block destruction after each hit.

Basic Power-ups

- At least one power-up implemented: “width+,” temporarily increasing paddle size.
- Power-ups fall after block destruction and can be collected by the paddle.

Stable Collisions

- All impacts (ball-paddle, ball-block, ball-wall) must be reliable and predictable, preventing the ball from “sticking” or passing through objects.

Optional Extensions

These features enhance learning and provide additional challenges:

- Level editor: manually create blocks to customize levels.
- Special blocks: explosive (affecting surrounding blocks) or indestructible.
- Varied power-ups: extra ball, slow ball, extra life.
- Hard mode: faster levels, more durable blocks, or challenging layouts.
- Advanced scoring system: multipliers, combos, and local rankings.

Learning Objectives

Upon completing this project, the developer should be able to:

Player Input

- Read horizontal inputs (keyboard or joystick) and move objects smoothly.

Basic 2D Physics and Multiple Collisions

- Detect collisions between the ball, paddle, blocks, and walls.
- Implement realistic and consistent bounces.

Level and Progression Management

- Control multiple levels and reset blocks between levels.

Power-ups and Special Interactions

- Generate random items, apply temporary effects, and remove them when necessary.

Visual Systems and Feedback

- Dynamically update graphical elements (block destruction, visual power-ups).

Optional Extensions

- Level editor and special blocks allow practice in modular architecture and code organization.

This project provides a solid foundation for understanding complex collisions, managing multiple objects, and enabling player interaction with the environment—essential concepts for any advanced 2D game development, including platformers, shooters, and puzzle games.

3 Exact Copy of Snake

General Description

This project consists of recreating the classic Snake game, where the player controls a snake moving within a grid or confined area. The snake grows as it consumes items

(usually “apples”) that appear randomly on the screen. The game ends when the snake collides with itself or, in some modes, with walls or obstacles.

The main goal of the project is to learn to handle grid-based movement, dynamic object growth, collision detection with multiple elements, and score management, consolidating fundamental concepts such as game logic, state updates per tick, and player input–based events.

Genre

Arcade / Grid-based

Typical features: sequential, discrete movements, reflex- and route-planning challenges, progressively increasing difficulty.

Difficulty

2/10 for solo implementation.

Simple to implement, but allows for extensions that increase complexity, such as dynamic obstacles, special-effect items, or infinite modes.

Game Modes

- **Standard Solo:** The player controls the snake and tries to achieve the highest score possible before colliding.

Optional Extensions:

- Infinite Mode: no wall limits; the snake can reappear from the opposite side.
- Dynamic Wall Challenges: obstacles that move or appear during gameplay.
- Special-effect Items: speed up or slow down the snake, reduce its size, etc.

Minimum Deliverables

To consider the project complete, the following elements must function correctly:

Snake Movement

- Movement based on regular ticks (consistent time intervals).
- Movement restricted to the grid or play area.
- Responsive control via keyboard or defined player input.

Apples or Growth Items

- Appear at valid random positions (not overlapping the snake).
- Eating an item increases the snake’s length.

Collision Detection

- The game ends if the snake collides with itself.
- Collisions with walls either end the game or follow defined rules.

Score

- Automatically updates as the snake consumes items.

- Clear score display on the screen.

Game Over

- Display or state indicating the game has ended.
- Ability to restart the game without glitches.

Optional Extensions

- Dynamic walls or moving obstacles: increases challenge and allows study of advanced collision logic.
- Infinite/wrap-around mode: the snake reappears on the opposite side when exiting the screen, requiring movement logic adjustments.
- Special-effect items: speed boost, slow down, shrink snake, award extra points.
- Increasing speed: gradually increases difficulty as the snake grows.
- Local multiplayer: two snakes competing on the same grid.

Learning Objectives

Upon completing this project, the developer should be able to:

Player Input

- Read and process discrete movements on a grid.

Tick-Based Movement

- Update snake position at regular intervals, maintaining game consistency.

Dynamic Object Growth

- Add segments to the snake when consuming items while maintaining proper collision logic.

Collision Detection and State Management

- Detect collisions with itself, walls, and items.
- Update game state accordingly (continue, Game Over).

Score Management and Basic UI

- Update the score in real time and display relevant player information.

Optional Extensions

- Wrap-around logic, special-effect items, and dynamic obstacles reinforce modular thinking, dynamic data structures, and event-driven programming.

This project is excellent for consolidating concepts of discrete movement, real-time entity growth, and event management, providing a solid foundation for more complex projects such as strategy games, puzzles, or grid-based simulations.

4 Exact Copy of Flappy Bird

General Description

This project consists of recreating the game Flappy Bird, a fast-paced arcade classic where the player controls a bird moving horizontally and must navigate gaps between pipes or vertical obstacles. The player interacts with the game by pressing a button to make the bird "flap," while gravity constantly pulls the character downward. Each successful passage through a set of pipes increases the score.

The main goal of the project is to teach basic physics applied to games, responsive single-button input, procedural obstacle generation, and incremental scoring logic, allowing the developer to consolidate essential 2D game fundamentals with runner mechanics.

Genre

Arcade / One-Tap Runner

Typical features: fast-paced matches, simple controls, difficulty based on precision and timing, immediate visual and audio feedback.

Difficulty

2/10 for solo implementation.

The project is relatively simple but allows for extensions that increase complexity, such as advanced physics, environmental variations, and replays.

Game Modes

- **Solo:** The player controls the bird and tries to achieve the highest score possible.

Optional Extensions:

- Infinite mode with difficulty variants: narrower obstacles or reduced distances between pipes.
- Ghost replays: show the trajectory of previous runs.
- Day/night mode: visual and environmental effects.

Minimum Deliverables

To consider the project complete, the following elements must function correctly:

Bird Movement

- Gravity constantly pulls the bird downward.
- Pressing the button makes the bird rise predictably.
- Smooth and responsive movement without glitches or inconsistent jumps.

Obstacles (Pipes)

- Deterministic or semi-random generation of gaps between pipes.
- Clear and reliable collisions between the bird and pipes or screen floor/ceiling.
- Removal of off-screen pipes to conserve resources.

Score

- Increment after passing each set of pipes.
- Clear, real-time display of the score.

Game Over

- The game ends immediately upon collision with obstacles.
- Final state displayed with the ability to restart without glitches.

Optional Extensions

- Day/night cycles: background or lighting visual changes.
- Wind or variable physics: altering speed or gravity for added challenge.
- Ghost replays: record and show player trajectory for analysis or competition.
- Power-ups: temporary invincibility, higher jumps, and more.
- Programmed challenges: pipes with movement patterns, progressive difficulty increase.

Learning Objectives

Upon completing this project, the developer should be able to:

Player Input

- Detect button presses and apply immediate effect on the character.

Simple 2D Physics

- Implement constant gravity and controlled jumps.
- Adjust speed and acceleration predictably.

Collision Detection

- Accurate collision between the bird, pipes, and screen boundaries.

Procedural Obstacle Generation

- Create semi-random gap patterns to maintain challenge and variety.

Score and Interface Management

- Dynamically update the score and display relevant player information.

Optional Extensions

- Implementing visual effects, additional physics, or replays allows practice in modularity, state management, and complex environment interaction.

This project is excellent for understanding simple player interaction with physics, obstacle generation, and immediate feedback—fundamentals applicable to countless platformers, runners, and arcade games.

5 Exact Copy of Space Invaders

General Description

This project consists of recreating the classic Space Invaders, an arcade shooter where the player controls a spaceship at the bottom of the screen and must destroy waves of aliens that progressively descend. The aliens move laterally in formation and advance toward the player, who must dodge enemy shots and fire back. Some versions include barriers that provide temporary protection for the player's ship.

The main goal of the project is to teach concepts of group movement, shooting and collisions, life management, and difficulty progression, while reinforcing 2D game logic fundamentals and enemy loop structures.

Genre

Shooter / Fixed Shooter (Gallery Shooter)

Typical features: restricted player movement, enemy waves with predictable patterns, fast reflexes, and management of shooting and survival.

Difficulty

3/10 for solo implementation.

Although simple, the project involves enemy formation logic, movement acceleration, and multiple interactive objects, making it a good next step after games like Pong or Breakout.

Game Modes

- **Solo:** The player controls the ship and faces waves of aliens.

Optional Extensions:

- Local versus or coop: two players facing aliens together or competitively.
- Infinite or survival modes: continuous waves of enemies with no defined limit.

Minimum Deliverables

To consider the project complete, the following elements must function correctly:

Player Movement

- Responsive horizontal movement at the bottom of the screen.
- Projectile firing that collides correctly with enemies.

Alien Waves

- Consistent lateral sweep pattern.
- Final acceleration when few enemies remain, increasing challenge.
- Aliens gradually descend, approaching the player.

Enemy Shots

- Projectiles fired by aliens can hit the player or barriers.
- Reliable collision detection and removal of off-screen projectiles.

Protective Barriers

- Barriers provide partial defense and can be destroyed by shots.
- Consistent and predictable behavior.

Lives and Game Over

- Player starts with 3 lives.
- Game Over occurs when all lives are lost or aliens reach the base.

Simple Boss

- One enemy or mini-boss at the end of the wave to add challenge and practice with enemy behavior variations.

Optional Extensions

- Power-ups: extra life, double shot, or temporary shield.
- Formation variations: different movement patterns for aliens.
- Barrier-free mode: increases difficulty and requires precision.
- Advanced enemy behavior: random shots, zig-zag movement, variable speed.
- Visual and audio effects: explosions, animations, and sound feedback for immersion.

Learning Objectives

Upon completing this project, the developer should be able to:

Player Input

- Horizontal movement and projectile firing with immediate response.

Group Enemy Movement

- Implement formation patterns and synchronized lateral movement.
- Adjust acceleration as the number of enemies decreases.

Shooting and Collisions

- Detect collisions between projectiles and enemies or barriers.
- Remove objects from the screen when necessary.

Life Management and Game Over

- Control the number of lives, restart, or end the game.

Bosses and Behavior Variation

- Introduce enemies with differentiated behavior to challenge the player.

Optional Extensions

- Practice modularity, power-up generation, and advanced enemy patterns.

This project is ideal for consolidating skills in 2D shooter games, synchronizing multiple enemies, collision detection, and game state management, preparing the developer for more complex shooters, tower defense, or real-time strategy games.

6. Exact Copy of Tetris

General Description

This project consists of faithfully recreating the classic Tetris, a timeless puzzle game based on fitting pieces called tetrominoes that continuously fall from the top of the screen. The player must rotate and move these pieces sideways to complete full horizontal lines, which are then cleared, freeing space. The game speed increases as progress is made, requiring quick reflexes and strategy.

The main goal of this project is to consolidate concepts such as matrix manipulation, rotation systems, grid-based collision detection, and game state management.

Genre

Puzzle / Stacking

A genre that requires spatial logic, planning, and reaction time.

Difficulty

3/10

While the rules are simple, the challenge lies in the correct implementation of collision systems, piece rotation, and line-clearing logic. It is an accessible project but already demands attention to code organization.

Game Modes

- **Solo (classic):** The player tries to survive as long as possible.

(Optional extensions may include additional modes such as versus or time attack, but the initial focus is on solo mode.)

Minimum Deliverables

To be considered complete, the game must include:

- **Seven Official Tetrominoes (I, O, T, S, Z, J, L),** with correct rotation behaviors.
- **Grid Collision System:** prevent pieces from passing through walls or already fixed blocks.
- **Line Clearing Logic:** complete horizontal lines disappear, and the blocks above fall down.
- **Scoring:** award points for each cleared line (bonus for multiple lines at once).
- **Speed Progression:** the game accelerates as the level increases.
- **Hold System:** allow the player to store one piece for later use.
- **Game Over Screen:** the game ends when there is no room for new pieces.

Optional Extensions

To expand learning and creativity, you can add:

- **Alternative Modes:**
    - Marathon: survive as long as possible until a time or score limit.

    - Sprint (40 lines): clear 40 lines in the shortest possible time.
    - Versus with Garbage: compete against another player by sending garbage lines.
- **Audio and Iconic Music:** reinforce the game's classic atmosphere.
- **Smooth Animations:** visual transitions when clearing lines.
- **Preview System:** show the next 3 or more pieces.
- **Local Leaderboard:** ranking of the highest scores.
- **Custom Skins:** change the look of tetrominoes and the board.

Learning Objectives

By completing this project, the student will have developed skills in:

- **Data Structures:** using matrices to represent the board and pieces.
- **Grid Collision Detection:** checking if a piece can move or rotate without exceeding boundaries.
- **Rotation System:** implementing tetromino rotation rules.
- **Game Loop Management:** handling piece falling at regular intervals synchronized with player inputs.
- **State Management:** switching between active game, pause, and game over.
- **Simple UI Design:** displaying score, level, next piece, and held piece.
- **Code Scalability:** structuring the project for easy future extensions.

Despite its simplicity, this project is a true school of programming logic and system design in games. By the end, you will have recreated one of the greatest classics in video game history and learned solid fundamentals that apply to nearly every type of 2D game.

7 Exact Copy of Frogger

General Description

This project consists of recreating the classic Frogger, an arcade game where the player controls a character (usually a frog) that must cross busy streets and rivers using moving platforms (logs, rafts, etc.), avoiding obstacles such as cars, trucks, and water. The player must carefully plan their movement to safely reach the other side of the screen.

The main goal of this project is to teach movement coordination, management of multiple moving elements and collisions, while consolidating player input skills, simple physics, and level progression logic in 2D games.

Genre

Arcade / Crossing

Typical features: fast-paced action, predictable movement patterns, reflex-based gameplay, and strategic planning to navigate obstacles.

Difficulty

4/10

The project is more challenging than Pong or Breakout because it requires synchronization of multiple moving elements and collision management with different types of objects. It is still feasible for solo implementation but requires careful attention to the update logic of each lane and platform.

Game Modes

- **Standard Solo:** cross all lanes and reach the goal.

Optional Extensions:

- Timed: complete each level within a time limit.
- Alternate Routes: different paths with varying difficulty.
- Dynamic Environment or Weather: rain, wind, or additional moving obstacles.

Minimum Deliverables

To consider the project complete, the following elements must function correctly:

Player Movement

- Immediate response to keyboard or joystick input.
- Respect screen boundaries and restrict movement within each lane.

Predictable Obstacle Cycles

- Cars, trucks, or other vehicles move in consistent, repetitive patterns.
- Logs and floating platforms move horizontally across rivers, transporting the player if they are on them.

Multiple Lanes

- At least three different lanes combining streets and rivers.
- Each lane has defined speed and obstacle density.

Final Goal

- The player must reach the top goal without collisions.
- Collision with vehicles or falling into water results in Game Over or a round reset.

Optional Extensions

- Dynamic weather and environment: rain, wind, or variable lighting that affects gameplay.
- Alternate routes: shorter but more difficult paths, or longer and safer ones.
- Timed levels: time limits per level or per crossing.
- Power-ups: extra speed, temporary invincibility, or bonus points.
- Animations and visual effects: animated water, realistically moving cars and logs.
- Leaderboard: record best times and scores.

Learning Objectives

Upon completing this project, the developer should be able to:

Player Input and Grid-Based Movement

- Read inputs and update position accurately while respecting screen boundaries.

Movement and Synchronization of Multiple Objects

- Cars, trucks, and platforms move predictably and independently.
- The player interacts correctly with moving platforms (transported or carried).

Collision Detection and Game Over

- Collisions with vehicles or falling into water reset the round.

Level Management and Progression

- Set up multiple lanes, each with distinct speed and obstacles.

Optional Extensions

- Timing, alternate routes, and power-ups introduce concepts of challenge design, difficulty balancing, and code modularity.

This project is excellent for learning how to synchronize multiple moving elements, manage complex collisions, and structure levels with increasing difficulty—skills that apply to platformers, runners, and dynamic puzzle games.

8 Exact Copy of Asteroids

General Description

This project consists of recreating the classic Asteroids, an open-arena shooter game where the player controls a spaceship subject to inertial physics. The ship can rotate, accelerate, and fire projectiles to destroy asteroids that move in various directions. When destroyed, asteroids break into smaller pieces until they disappear completely. The screen uses wrap-around, meaning objects that exit one side reappear on the opposite side. Some levels may include UFOs that shoot at the player.

The main goal of this project is to teach concepts of 2D inertial physics, collision of multiple moving objects, and entity fragmentation, while consolidating fundamentals of input, movement, and open-arena design.

Genre

Shooter / Inertial Arena

Typical features: physics-controlled ship, precise shooting, free and dynamic movement, management of multiple moving targets.

Difficulty

4/10

The main challenge is implementing inertial physics, precise collisions, and asteroid fragmentation. Wrap-around logic requires careful implementation, but it is still feasible for solo development.

Game Modes

- **Standard Solo:** the player controls the ship and tries to survive by destroying all asteroids.

Optional Extensions:

- Wave-based modes: sequences of asteroids with increasing difficulty.
- Infinite arena: continuous asteroid generation.
- Leaderboard: local or global score ranking.
- Environmental hazards: additional meteors, black holes, or space storms.

Minimum Deliverables

To consider the project complete, the following elements must function correctly:

Ship Movement

- Applied inertial physics: acceleration and deceleration with momentum.
- Precise rotation and projectile firing.

Asteroids

- Linear movement with wrap-around on the screen.
- Fragmentation in 2–3 stages when hit.
- Correct collision detection with shots and the ship.

Projectiles

- Fired from the ship toward the target.
- Removed when colliding with asteroids or leaving the screen.

Occasional UFO

- Appears at random intervals or based on score.
- Shoots at the ship with simple patterns.

Scoring and Game Over

- Points awarded for destroying asteroids and UFOs.
- Game Over occurs when the ship collides with asteroids or enemy projectiles.

Optional Extensions

- Wave-based modes: each wave increases asteroid speed, quantity, and size.
- Environmental hazards: random meteors, black holes, gravity fields.
- Local or global leaderboard: record high scores.
- Power-ups: temporary shield, double shot, extra speed.

- Visual and audio effects: explosions, fragment animations, and classic soundtrack.
- Themed arenas: different backgrounds or fixed obstacles.

Learning Objectives

Upon completing this project, the developer should be able to:

2D Inertial Physics

- Apply acceleration, deceleration, and momentum to the ship.
- Implement screen wrap-around for all moving objects.

Player Input and Precise Control

- Rotate, accelerate, and shoot with immediate response.

Collision of Multiple Moving Objects

- Detect impacts between the ship, projectiles, asteroids, and UFOs.

Fragmentation and Dynamic Object Generation

- Split asteroids into smaller pieces while maintaining consistent movement.

Scoring and Game Over Management

- Control the ship's life, scoring, and game states.

Optional Extensions

- Implementing waves, environmental hazards, power-ups, and leaderboards develops modularity, scalability, and progressive challenge design.

This project is ideal for practicing free movement with realistic physics, managing multiple objects with complex collisions, and fragmentation dynamics—skills that apply to shooters, space simulators, and action games with advanced physics.

9 Exact Copy of Pac-Man

General Description

This project consists of recreating the classic Pac-Man, an arcade game in which the player controls a character moving through a maze to collect pellets while avoiding ghosts. In addition to regular pellets, there are power pellets that temporarily make ghosts vulnerable, allowing the player to chase and eliminate them for extra points. The goal is to collect all pellets in the maze to advance to the next level while managing lives and avoiding being caught.

The main focus of this project is to teach grid-based movement, simple enemy AI, collision and item collection, as well as score, life, and level progression management.

Genre

Arcade / Maze

Typical features: maze navigation, enemy evasion, item collection, timing-based challenges, and predictable AI patterns.

Difficulty

5/10

This project is more complex due to ghost AI, multiple interactive objects, and the need for scoring and lives systems. Solo implementation requires careful planning of logic and precise event control.

Game Modes

- **Standard Solo:** control Pac-Man through the maze, collecting all pellets.

Optional Extensions:

- Turn-based multiplayer: two players alternating turns or controlling different characters.
- Variant mazes: multiple maps with different challenges.
- Timed or score-based mode: complete mazes quickly or maximize points.

Minimum Deliverables

To consider the project complete, the following elements must function correctly:

Player Movement

- Responsive control within the maze.
- Movement restricted to the maze grid.

Ghost AI

- Classic patterns: direct chase, prediction, patrol, or fleeing.
- Behavior change when Pac-Man consumes a power pellet.

Pellet and Power Pellet Collection

- Pellets disappear when collected.
- Power pellets temporarily activate ghost vulnerability.

Scoring and Game Over

- Points awarded for pellets, power pellets, and consumed ghosts.
- Game Over occurs when all player lives are lost.

Level Progression

- Completing all pellets advances to the next maze.
- Speed or difficulty gradually increases.

Optional Extensions

- Bonus fruits: items appear periodically, granting extra points.
- Variant mazes: multiple layouts and challenges.
- Alternating or simultaneous multiplayer: two players competing or taking turns.

- Animations and sound effects: visual and audio feedback for collection, power pellets, and captures.
- Advanced AI: ghosts with more sophisticated strategies or adaptive behavior.
- Leaderboard: track high scores.

Learning Objectives

Upon completing this project, the developer should be able to:

Grid-Based Movement and Player Input

- Navigate precisely through a maze with strict boundaries.

Classic Enemy AI

- Implement distinct movement patterns and temporary behavior changes.

Collision Detection and Item Collection

- Detect collisions with ghosts or collection of pellets.

Score and Lives Management

- Correctly increment score and manage Game Over states.

Level Progression

- Transition between mazes with gradual difficulty increase.

Optional Extensions

- Adding bonus items, variant mazes, or multiplayer exercises modularity and challenge design.

This project is excellent for learning to combine grid-based movement, AI logic, and item collection—skills applicable to maze games, puzzles, and top-down adventure games.

10 Exact Copy of Galaga

General Description

This project consists of recreating the classic Galaga, a vertical gallery shooter in which the player controls a ship at the bottom of the screen. The goal is to destroy waves of flying enemies that appear in pre-defined formations and attack in patterned movements. Some enemies can capture the player's ship, increasing the challenge. The player must avoid enemy fire and eliminate all aliens to advance to the next stage.

The main objective of this project is to teach movement in pre-defined patterns, management of multiple enemies, collision detection, and continuous shooting logic, while reinforcing concepts of stage progression and scalable challenge design.

Genre

Shooter / Vertical Gallery

Typical features: player-controlled ship at the bottom of the screen, waves of enemies, predictable attacks, and increasing difficulty per stage.

Difficulty

3/10

The project requires coordination of multiple moving objects and projectiles, but the movement and collision logic is relatively simple, making it feasible for solo implementation.

Game Modes

- **Standard Solo:** the player faces waves of enemies and tries to survive as long as possible.

Optional Extensions:

- Local Versus: two players on split screens or alternating turns.
- Wave Modes: each stage increases the number and speed of enemies.
- Infinite Survival: continuous enemy waves until all lives are lost.

Minimum Deliverables

To consider the project complete, the following elements must function correctly:

Player Ship Movement

- Responsive horizontal control.
- Continuous or dual firing.

Enemies

- Movement in pre-defined formations and curved trajectories during attacks.
- Enemy shots with collision detection.

Stages

- At least 5 distinct stages, each with gradually increasing difficulty.
- Ability to restart a stage after partial Game Over.

Ship Capture

- Enemies can temporarily capture the player’s ship, increasing risk.

Scoring and Game Over

- Points awarded for each enemy destroyed.
- Game Over occurs when all lives are lost.

Optional Extensions

- Dual Capture: enemies can capture multiple ships, requiring rescue strategies.
- Bosses: stronger enemies with special attacks at the end of stages.
- Double Power-ups: faster shots, temporary shields, or multiple projectiles.
- Animations and sound effects: explosions, visual shots, and iconic sounds.

- Leaderboard: track high scores.
- Themed stages: different attack patterns or enemies per stage.

Learning Objectives

Upon completing this project, the developer should be able to:

Player Movement and Input

- Control the ship precisely and fire projectiles.

Enemy Movement in Formation

- Create consistent attack patterns and curved trajectories.

Collision Detection

- Detect collisions between player shots and enemies, and enemy shots and the player.

Stage and Difficulty Management

- Implement increasing speed, enemy count, and attack complexity.

Ship Capture

- Handle risk escalation and possible rescue strategies.

Optional Extensions

- Implementing bosses, power-ups, and advanced modes exercises modularity, challenge design, and code scalability.

This project is ideal for learning patterned movement, managing multiple moving objects, and continuous shooting logic—essential skills for arcade shooters and 2D action games with progressive difficulty.

11 Exact Copy of Donkey Kong

General Description

This project consists of recreating the classic Donkey Kong, a platformer in which the player controls a character who must climb vertical platforms, avoid obstacles such as rolling barrels, and reach the top to rescue a character (Pauline). The game involves lateral and vertical movement, jumping, ladder climbing, and interaction with objects such as hammers that temporarily allow destroying barrels.

The main objective of the project is to teach concepts of platform movement, collision with dynamic obstacles, management of multiple levels and power-ups, while consolidating player input skills and level design with progressive challenge.

Genre

Platformer / Puzzle-Arena

Typical features: vertical platforms, moving obstacles, ladder climbing, timing challenges, and temporary item collection.

Difficulty

4/10

Requires implementation of simple jump physics, collision with multiple objects, and logic for progression across screens, making it challenging but feasible for solo implementation.

Game Modes

- **Standard Solo:** player controls the main character to rescue Pauline.

Optional Extensions:

- High Scores: record maximum score per stage or time.
- Difficulty Variants: more barrels, moving platforms, or additional challenges.

Minimum Deliverables

To consider the project complete, the following elements must function correctly:

Player Movement

- Precise lateral movement and realistic jumps.
- Ladder climbing between platforms.

Dynamic Obstacles

- Rolling barrels on predictable paths with collision detection.
- Interaction with hammers as temporary power-ups that allow destroying barrels.

Levels / Screens

- At least 4 different screens with progressively increasing challenges.

Final Objective

- Rescue Pauline at the top of the screen.

Scoring and Game Over

- Points awarded for destroying barrels, collecting power-ups, and completing levels.
- Game Over occurs when the player loses all lives.

Optional Extensions

- Fire: additional enemies moving along platforms.
- Barrel Variants: barrels with different behaviors, such as bouncing or splitting.
- High Scores: local or global leaderboard to track scores.
- Visual and Sound Effects: animations for barrels, power-ups, and point collection.
- Alternative Modes: timed stages, speed challenges, or multiple objectives.

Learning Objectives

Upon completing this project, the developer should be able to:

Player Platform Movement

- Implement lateral running, jumping, and ladder climbing.

Dynamic Obstacle Management

- Rolling barrels and precise collision detection with the player.
- Use of power-ups that temporarily alter gameplay.

Level Design and Progression

- Create progressive screens with increasing difficulty.

Scoring and Game Over

- Correctly increment points and manage lives and Game Over states.

Optional Extensions

- Adding extra enemies, visual effects, or a leaderboard exercises modularity, scalability, and challenge design.

This project is excellent for learning platform movement, collision with dynamic obstacles, and progressive level design, fundamental skills for both classic and modern platform games.

12 Exact Copy of Q*bert

General Description

This project consists of recreating the classic Q*bert, a puzzle and isometric platformer in which the player controls a character that jumps between cubes arranged in a pyramid shape. Each cube changes color when the character jumps on it, and the goal is to change the color of all cubes to complete the level. The player must avoid hopping enemies that move across the pyramid and obstacles that can cause falls or Game Over.

The main objective of this project is to teach isometric movement, collision, and object state-changing logic, while consolidating concepts of level design and risk management in platform games.

Genre

Puzzle / Isometric Platformer

Typical features: jump-based movement, block state-changing logic, enemies following simple patterns, focus on planning and timing.

Difficulty

3/10

The project requires understanding isometric coordinates and block state logic but is relatively simple for solo implementation.

Game Modes

- **Standard Solo:** player completes the pyramid by changing all cube colors while avoiding enemies.

Optional Extensions:

- Levels with variant pyramids: different sizes or additional blocks.
- Timed modes: complete levels within a time limit.
- Leaderboard: track score by completed levels or time.

Minimum Deliverables

To consider the project complete, the following elements must function correctly:

Pyramid and Cubes

- 5x5 cube structure in isometric perspective.
- Cubes change color when the player jumps on them.

Player Movement

- Jumps between cubes respecting isometric geometry.
- Responsive control allowing anticipation of movements.

Enemies and Obstacles

- Enemies hop or follow simple patterns that can capture the player.
- Falls or collisions result in loss of life or Game Over.

Scoring and Game Over

- Completing the pyramid awards points.
- Game Over occurs when all lives are lost.

Optional Extensions

- Saver discs: special blocks that prevent falls or allow escape from enemies.
- Pyramid variations: larger pyramids with additional colors or different shapes.
- Extra disc: bonus items granting extra points or temporary abilities.
- Animations and visual effects: highlight altered blocks, jump animations, and enemy movement.
- Leaderboard and timed challenges: ranking by time or maximum score.

Learning Objectives

Upon completing this project, the developer should be able to:

Isometric Movement

- Perform precise jumps between blocks and correctly calculate positions.

Object State Management

- Cubes change color when touched by the player.

Enemy and Obstacle Management

- Implement enemies with simple movement patterns.
- Detect collisions and falls, triggering Game Over.

Level Design and Progression

- Create progressive challenges by increasing pyramid complexity.

Optional Extensions

- Adding saver discs, variant pyramids, and bonus items exercises modularity, challenge design, and code scalability.

This project is excellent for learning isometric movement, block state logic, and strategic level design, skills that apply to puzzles, platformers, and visually oriented reasoning games.

13 Exact Copy of Dig Dug

General Description

This project consists of recreating the classic Dig Dug, an action game where the player controls a character who digs underground tunnels to confront enemies. The player can defeat enemies in two ways: inflating them until they explode or dropping rocks on them. The game also features bonus items, such as vegetables that increase the score.

The main objective of this project is to teach concepts of grid-based underground movement, enemy collision, basic physics of falling objects, and attack mechanics, while consolidating player input handling, scoring logic, and progressive level design skills.

Genre

Action / Digging

Typical features: underground exploration, simple enemy AI, dynamic obstacles, item collection, and strategic combat.

Difficulty

4/10

This project requires implementing movement in a confined space, managing enemies, detecting collisions, and applying basic physics for falling objects. It is more challenging than simple platformers but feasible for solo implementation.

Game Modes

- **Standard Solo:** player faces enemies, digs tunnels, and collects bonuses.

Optional Extensions:

- Stage-based modes: each level with different layouts and increasing difficulty.
- Infinite survival: continuous waves of enemies until all lives are lost.
- Leaderboard: track high scores per level or total.

Minimum Deliverables

To consider the project complete, the following elements must function correctly:

Player Movement

- Digging in all directions within the terrain.
- Respect for tunnel boundaries and natural obstacles.

Enemies

- At least two types of enemies with basic movement and predictable patterns.
- Ability to defeat enemies by inflating them.

Tunnels and Collisions

- Tunnels are diggable and restrict movement of both player and enemies.
- Collision detection between player, enemies, and environmental objects.

Bonus Items

- Vegetables that appear periodically and increase score when collected.

Scoring and Game Over

- Points correctly awarded for defeating enemies and collecting bonuses.
- Game Over occurs when all lives are lost.

Optional Extensions

- Rocks: objects that can be dropped on enemies, causing area destruction.
- Evasive AI: enemies that flee the player or change behavior based on proximity.
- Stage-based modes: variable layout and number of enemies per level.
- Additional power-ups: items that increase speed, strength, or attack range.
- Animations and visual effects: highlight enemy inflation, rock falls, and vegetable collection.
- Leaderboards and timed challenges: ranking by score or time to complete a level.

Learning Objectives

Upon completing this project, the developer should be able to:

Grid-based Movement and Digging

- Create tunnels and allow the player to navigate them smoothly.

Enemy Interaction

- Implement basic enemy behavior, player attacks, and collision detection.

Dynamic Objects

- Simulate simple physics for rocks or obstacles that can fall on enemies.

Scoring and Game Over

- Increment points correctly and manage player lives.

Level Design and Progression

- Create varied tunnel layouts and progressively increase the challenge.

Optional Extensions

- Implementing evasive AI, rocks, and power-ups exercises modularity, challenge design, and code scalability.

This project is ideal for learning underground grid-based movement, strategic combat, and interactive level design—skills applicable to action, strategy, and dynamic puzzle games.

14 Exact Copy of Centipede

General Description

This project consists of recreating the classic Centipede, a vertical shooter in which the player controls a character at the bottom of the screen, shooting at a descending centipede that moves among scattered mushrooms. Each segment of the centipede must be destroyed individually, and mushrooms alter the centipede's path, creating strategic challenges. The goal is to eliminate all enemies, score points, and survive as many waves as possible.

The main objective of this project is to teach segmented enemy movement, collision detection, interaction with dynamic obstacles, and score management, consolidating concepts of 2D shooters and wave-based progression logic.

Genre

Shooter / Vertical Gallery

Typical features: descending enemies, obstacles that alter trajectories, continuous player shooting, focus on reflexes and strategy.

Difficulty

4/10

The project requires coordination of multiple moving elements (centipede, mushrooms, projectiles) and precise collision control. It is challenging but feasible for solo implementation.

Game Modes

- **Standard Solo:** player faces the descending centipede and other enemies, trying to survive and accumulate points.

Optional Extensions:

- Infinite waves: continuous enemy waves for survival testing.
- Timed modes: complete a set number of waves within a time limit.
- Leaderboard: track highest score achieved.

Minimum Deliverables

To consider the project complete, the following elements must function correctly:

Segmented Centipede

- Segments move in descending patterns among mushrooms.
- Each segment can be destroyed individually.

Player Projectiles

- Responsive vertical shooting with collision detection against centipede segments and mushrooms.

Mushrooms

- Obstacles that alter the centipede's trajectory and can be destroyed or remain fixed.

Scoring and Lives

- At least 3 lives for the player.
- Points correctly awarded for destroying centipede segments and mushrooms.
- Game Over occurs when all lives are lost.

Optional Extensions

- Variant insects: additional enemies with different movement patterns.
- Power-ups: double shots, increased speed, or temporary shields.
- Infinite waves: progressively harder waves with more enemies or higher speed.
- Animations and visual effects: highlight segment destruction, projectiles, and mushroom interaction.
- Leaderboards and timed challenges: ranking by highest score or survival time.

Learning Objectives

Upon completing this project, the developer should be able to:

Segmented Enemy Movement

- Implement a centipede with independent segments moving in descending patterns.

Interaction with Dynamic Obstacles

- Detect collisions with mushrooms and adjust enemy trajectories.

Player Control and Shooting

- Create responsive vertical shots and collision detection with enemies and obstacles.

Score and Game Over Management

- Increment points for destroyed segments and mushrooms.
- Manage lives and Game Over states.

Optional Extensions

- Implementing additional enemies, power-ups, and infinite waves exercises modularity, scalability, and progressive challenge design.

This project is ideal for learning 2D shooter mechanics with segmented enemies, dynamic obstacle logic, and progressive wave design—skills applicable to action games, arcade shooters, and strategic reflex challenges.

15 Exact Copy of Defender

General Description

This project consists of recreating the classic Defender, a horizontal side-scrolling shooter in which the player controls a spaceship that must protect humans on the ground from alien invaders. The player can shoot enemies, rescue captured humans, and use hyperspace as an emergency tool. The action takes place in a continuous horizontally scrolling environment, requiring attention to both enemies and the humans' positions.

The main objective of this project is to teach continuous lateral movement, collision detection with multiple elements, NPC rescue mechanics, and risk/reward gameplay, consolidating skills in 2D shooters, side-scrolling, and managing multiple simultaneous objectives.

Genre

Shooter / Side-scrolling

Typical features: horizontal scrolling, varied enemies, secondary objectives (rescue), focus on reflexes and attack/defense strategy.

Difficulty

5/10

The project requires coordination between multiple elements: enemies, rescuable humans, player projectiles, and hyperspace, making it more complex than simple vertical or arcade shooters, but still feasible for solo implementation.

Game Modes

- **Standard Solo:** player controls the ship to eliminate enemies and rescue humans.

Optional Extensions:

- Timed or wave-based modes: complete a set number of enemy waves within a time limit.
- Infinite survival: withstand continuous waves of enemies.
- Leaderboard: track maximum scores for survival, rescues, or enemies destroyed.

Minimum Deliverables

To consider the project complete, the following elements must function correctly:

Player Movement

- Ship moves horizontally with continuous screen wrapping.
- Functional use of hyperspace for emergencies.

Enemies

- At least 4 enemy types with varied movement and attack patterns.
- Interaction with player projectiles and the ability to destroy or capture humans.

Rescuable Humans

- NPCs that can be captured by enemies and rescued by the player.
- Successful rescues yield additional points.

Scoring and Lives

- At least 3 lives for the player.
- Points correctly awarded for enemies destroyed and humans rescued.
- Game Over occurs when all lives are lost or all humans are eliminated.

Optional Extensions

- Treasure chests: bonus items that appear randomly to increase score.
- Extra ships: additional enemies or allies with unique behaviors.
- Variant planets: different terrains or obstacles affecting movement and strategy.
- Temporary power-ups: increased speed, double shots, or shields.
- Visual and animation effects: highlight enemy destruction, human rescues, and hyperspace use.

Learning Objectives

Upon completing this project, the developer should be able to:

Lateral Movement and Side-scrolling

- Implement continuous horizontal scrolling and precise ship navigation.

Management of Multiple Simultaneous Elements

- Handle enemies with simple AI, rescuable humans, and player projectiles.
- Detect collisions among all these elements.

Special Resources and Risk/Reward Mechanics

- Use hyperspace strategically for survival.
- Rescues increase points and require constant attention.

Scoring and Game Over

- Increment points correctly for enemies destroyed and humans rescued.
- Manage player lives and Game Over states.

Optional Extensions

- Implementing power-ups, additional enemies, and variable terrains exercises modularity, challenge design, and code scalability.

This project is ideal for learning 2D side-scrolling shooter mechanics with multiple objectives, NPC interaction, and risk management—skills applicable to strategic action games and modern shooters.

16 Exact Copy of Scramble

General Description

This project consists of recreating the classic Scramble, a horizontal side-scrolling shooter in which the player controls a spaceship advancing through a continuously moving environment. The player must avoid or destroy enemies, neutralize ground-based bombs, and manage fuel to survive. The combination of shooting and survival elements creates a challenge that requires both reflexes and planning.

The main objective of this project is to teach continuous movement, side-scrolling, collision detection with enemies and obstacles, resource management (fuel), and scoring, consolidating skills in 2D shooters and progressive level design.

Genre

Shooter / Side-scrolling

Typical features: horizontal scrolling, terrain obstacles, resource management (fuel), focus on reflexes and strategy.

Difficulty

4/10

The project involves continuous lateral movement, targeted shooting, collisions with multiple elements, and fuel management, making it a moderate challenge but feasible for solo implementation.

Game Modes

- **Standard Solo:** player advances horizontally, destroys enemies and bombs, and manages fuel.

Optional Extensions:

- Timed or level-based modes: complete certain sections before fuel runs out.
- Infinite survival: continuous environment with gradually increasing difficulty.
- Leaderboard: track high scores for distance traveled or enemies destroyed.

Minimum Deliverables

To consider the project complete, the following elements must function correctly:

Continuous Horizontal Scrolling

- Environment moves smoothly, simulating spaceship flight.

Player Movement and Shooting

- Ship responds to horizontal and vertical input.
- Functional shooting against aerial enemies and ground bombs.

Enemies and Obstacles

- Ground bombs that can destroy the ship if not avoided or destroyed.
- At least simple aerial enemies representing a constant threat.

Scoring and Resources

- Points correctly awarded for destroying enemies and obstacles.
- Basic fuel management (decreases over time or action, without instant depletion).
- Game Over occurs when all lives are lost or fuel runs out.

Optional Extensions

- Fuel pickups: items appearing along the way to extend survival.
- Additional ground enemies: varied types of bombs or ground vehicles.
- High scores: ranking by points or distance traveled.
- Temporary power-ups: speed boost, multiple shots, or shields.
- Visual and audio effects: highlight enemy destruction, explosions, and fuel collection.

Learning Objectives

Upon completing this project, the developer should be able to:

Continuous Lateral Movement (Side-scrolling)

- Implement a continuously moving environment and coordinate player ship movement.

Management of Multiple Elements

- Handle aerial enemies and ground bombs with collision detection.
- Strategic combat with targeted shooting.

Resource Management

- Control fuel, providing tension and strategic decision-making for the player.

Scoring and Game Over

- Increment points correctly and manage player lives.
- Game Over conditions: loss of all lives or depletion of fuel.

Optional Extensions

- Implementing power-ups, enemies, and pickups exercises modularity, challenge design, and code scalability.

This project is ideal for learning 2D side-scrolling shooter mechanics with resource management, ground obstacles, and attack strategy—skills applicable to action games and classic shooters.

17 Exact Copy of Tempest

General Description

This project consists of recreating the classic Tempest, an arcade shooter with a tubular tunnel in simulated 3D perspective (2D with a sense of depth). The player controls a flipper that moves around the edge of the tunnel, shooting at waves of enemies descending through the tube. Each level features enemy patterns advancing at different speeds, creating reflex and precision challenges.

The main objective of this project is to teach simulated 3D perspective in 2D, circular movement, targeted shooting, and enemy wave logic, consolidating arcade shooter concepts and progressive challenge design.

Genre

Shooter / Tubular (Arcade)

Typical features: simulated perspective, circular player movement, enemy waves, fast reflexes, and shooting strategy.

Difficulty

5/10

Implementing the pseudo-3D perspective and circular movement increases complexity, requiring care with graphic transformations, collisions, and enemy patterns, but it is still feasible for solo implementation.

Game Modes

- **Standard Solo:** player controls the flipper and eliminates waves of enemies inside the tunnel.

Optional Extensions:

- Level-based modes: different tunnel layouts and enemy speeds.
- Infinite survival: continuous waves with increasing difficulty.
- Leaderboard: maximum score per level or survival.

Minimum Deliverables

To consider the project complete, the following elements must function correctly:

Simulated Perspective (Pseudo-3D)

- Tubular tunnel represented in 2D with a sense of depth.
- Player movement around the tunnel edge.

Flipper and Controls

- Flipper responds to circular player input.
- Shooting that hits enemies along the tunnel segments.

Enemy Waves

- Basic enemy patterns descending through the tunnel.
- Correct collision detection between shots and enemies.

Scoring and Game Over

- Points correctly awarded for enemy elimination.
- Game Over occurs when the player is hit or enemies reach the end of the tunnel.

Optional Extensions

- Jumpers: enemies that skip segments or appear unexpectedly.
- Tracers: enemies that intelligently follow the player.
- Level-based modes: different tunnel layouts, speeds, and enemy densities.
- Power-ups and bonuses: multiple shots, temporary shields, or speed boosts.
- Visual and audio effects: highlight shooting, explosions, and circular movement.

Learning Objectives

Upon completing this project, the developer should be able to:

Simulated 3D Perspective in 2D

- Create a sense of depth in a tunnel using 2D coordinates and transformations.

Circular Player Movement

- Implement flipper control around the tunnel edge, ensuring precision in shooting.

Management of Enemy Waves

- Create descending movement patterns and collision detection with player shots.

Scoring and Game Over

- Increment points correctly and manage game state when the player is hit.

Optional Extensions

- Implement special enemies and different modes, exercising modularity, scalability, and progressive challenge design.

This project is ideal for learning arcade shooter mechanics with simulated perspective, circular movement, and enemy wave logic, skills applicable in advanced-action games and creative visual design.

18 Exact Copy of Robotron: 2084

General Description

This project consists of recreating the classic Robotron: 2084, a twin-stick arcade shooter in which the player controls a character in an open arena and must face crowds

of enemy robots while rescuing humans. The player moves the character with one stick (or set of keys) and fires independently in any direction with the other.

The main objective of this project is to teach multidirectional control, independent shooting, enemy horde logic, collision detection, and NPC rescue, consolidating top-down shooter skills with multitasking and fast reflexes.

Genre

Shooter / Twin-stick (Arcade)

Typical features: top-down view, movement independent from shooting, enemy hordes, NPC rescue, fast pace, and reflex-intensive gameplay.

Difficulty

5/10

Complexity comes from the need for independent movement and shooting in 8 directions, controlling multiple enemies simultaneously, and implementing NPC rescue logic.

Game Modes

- **Standard Solo:** player faces waves of enemies and rescues humans.

Optional Extensions:

- Local co-op: two players controlling characters simultaneously in the same arena.
- Endless survival: infinite waves with progressively increasing difficulty.
- Leaderboard: track high scores for survival or human rescues.

Minimum Deliverables

To consider the project complete, the following elements must function correctly:

Multidirectional Movement and Shooting

- Character control in 8 directions.
- Independent shots that can be fired in any direction, regardless of movement.

Enemies and Hordes

- Basic enemy robots that approach the player.
- Correct collision detection between shots and enemies, and between enemies and player.

Humans to Rescue

- NPCs that must be saved, adding points and strategic challenge.
- Not rescuing or losing NPCs should not freeze the game.

Scoring and Game Over

- Points correctly awarded for enemy kills and human rescues.
- Game Over occurs when all lives are lost.

Optional Extensions

- Grunts, hulks, and tanks: enemies with differentiated behavior and increasing difficulty.
- Local co-op: two simultaneous players in the arena, requiring coordination.
- Power-ups: multiple shots, increased speed, or temporary shields.
- Visual and audio effects: highlight shooting, explosions, and rescues.
- Leaderboard: track maximum score and rescue records.

Learning Objectives

Upon completing this project, the developer should be able to:

Independent Movement and Shooting

- Implement multidirectional control separate from shooting direction, simulating twin-stick gameplay.

Managing Multiple Enemies and NPCs

- Create enemy hordes with basic patterns and detect collisions with shots and player.
- Implement NPC rescue logic and score tracking.

Scoring and Game Over

- Increment points correctly and manage player lives.
- Game Over occurs upon losing all lives.

Optional Extensions

- Implementing advanced enemies and local co-op exercises modularity, scalability, and progressive challenge design.

This project is ideal for learning top-down twin-stick shooter mechanics, enemy hordes, and strategic NPC rescue, skills applicable to fast-paced action games and multitasking game design.

19 Exact Copy of Joust

General Description

This project consists of recreating the classic Joust, an action arena game in which players control knights mounted on ostriches. The goal is to knock down enemies by colliding with them using a lance, while staying airborne and avoiding environmental obstacles. The game features floating platforms, simulated gravity, and controlled flight mechanics.

The main objective of this project is to teach aerial movement with simple physics, directional collisions, interaction with platforms and enemies, and life management, consolidating concepts of 2D action games with arcade physics.

Genre

Action / Arena (Arcade)

Typical features: closed arena, flight mechanics, melee combat with collision, platforms, and continuous challenge elements.

Difficulty

4/10

Solo implementation requires handling controlled aerial movement, directional collisions, and basic gravity for ascending and descending, but remains accessible for individual study.

Game Modes

- Local versus: two players can compete simultaneously on the same screen.

Optional Extensions:

- Online multiplayer: network-based competition.
- Single-player mode against AI: enemies controlled by simple logic.
- Endless survival: successive waves of enemies.

Minimum Deliverables

To consider the project complete, the following elements must function correctly:

Controlled Aerial Movement

- Mounted knights can ascend and descend with simulated gravity.
- Floating and platform collision behave predictably.

Enemy Collisions and Lance Impact

- Determine collision direction: player wins if above the enemy.
- Proper enemy reset or loss of life upon defeat.

Platforms and Eggs

- Platforms are positioned stably.
- “Eggs” (items or defeated enemies) spawn correctly and can be collected.

Scoring and Lives

- Points increment correctly when defeating enemies.
- Each player has 3 lives.

Optional Extensions

- Pterodactyls: special enemies appear periodically, increasing difficulty.
- Rising lava: environmental element that forces strategic movement.
- Online multiplayer: enable network-based competition.
- Power-ups: temporary speed boost or attack range increase.

- Visual and audio effects: highlight lance collisions, falls, and egg collection.

Learning Objectives

Upon completing this project, the developer should be able to:

Aerial Movement and Simple Physics

- Implement ascending, descending, and simulated gravity for 2D platforms.

Directional Collisions and Combat Logic

- Determine the winner of a collision based on vertical position.
- Proper enemy reset or player life loss upon defeat.

Score and Life Management

- Increment points correctly for defeats and egg collection.
- Track number of lives and detect Game Over.

Optional Extensions

- Implement special enemies and environmental elements to exercise modularity, progressive challenge design, and arena balance.

This project is ideal for learning 2D arcade physics, directional collisions, arena combat, and platform design, skills applicable in multiplayer action games and fast-reflex challenges.

20 Exact Copy of Marble Madness

General Description

This project consists of recreating the classic Marble Madness, a precision racing game where the player controls a ball that must navigate mazes in an isometric perspective while avoiding falls and obstacles. The ball is affected by tilt physics, simulating acceleration, deceleration, and inertia as it rolls over inclined surfaces.

The main objective of this project is to teach 2D physics control simulating 3D, tilt-based movement, timer management, and challenging level design, consolidating skills in isometric racing puzzles.

Genre

Racing / Puzzle (Arcade)

Typical features: time-based racing, mazes with obstacles, precise movement physics, isometric perspective to simulate 3D.

Difficulty

5/10

Complexity arises from controlling the ball’s physics, simulating tilt, and implementing challenging obstacles, requiring careful attention to collisions and smooth movement.

Game Modes

- Standard solo: player navigates tracks aiming to complete them within minimum time.

Optional Extensions:

- Alternating multiplayer: two or more players compete in turns, recording best times.
- Practice mode: no timer, for training on specific tracks.
- Mode with dynamic hazards: moving platforms or obstacles requiring quick reflexes.

Minimum Deliverables

To consider the project complete, the following elements must function correctly:

Ball Movement and Tilt Physics

- The ball responds gradually to controls, with realistic acceleration and deceleration.
- Collisions with edges and obstacles prevent the ball from leaving the maze predictably.

Mazes and Tracks

- At least 3 different tracks with varied challenges.
- Paths and inclines are well-defined, allowing controlled navigation.

Timer and Scoring

- Accurate measurement of completion time.
- Victory or failure system based on time or successful maze completion.

Optional Extensions

- Dynamic obstacles: moving platforms, holes, ramps.
- Alternating multiplayer: time-based competition between players.
- Practice mode: training without time penalties.
- Leaderboards: ranking best times per track.
- Power-ups: temporary speed boost or ability to jump small obstacles.

Learning Objectives

Upon completing this project, the developer should be able to:

Physics-Based Control

- Implement gradual movement with acceleration, deceleration, and inertia.
- Detect collisions with obstacles and edges reliably.

Challenging Maze Design

- Create tracks with inclines and paths that test precision and reaction time.

- Implement checkpoints and restart logic in case the ball falls.

Time and Score Management

- Time runs and determine victory or failure.
- Record times for performance analysis.

Optional Extensions

- Implement hazards and alternating multiplayer to develop modularity, difficulty balancing, and competitive design.

This project is ideal for learning physics-based movement, isometric 2D mazes, and racing puzzles, skills applicable in precision games, physics platformers, and fast-reflex challenges.

Game Jam – Arcade Quickfire (48h)

General Description

This project proposes creating a simplified clone of a classic arcade game in just 48 hours, simulating a game jam experience. The goal is to produce a functional game with direct controls, short gameplay loops, a score system, and lives, using concepts already studied in previous core projects.

The main focus is on rapid implementation practice, refining basic mechanics, and making decisions under time constraints, consolidating the ability to create playable games in short periods.

Mandatory Basic Mechanics:

- Player movement via direct input (keyboard or joystick).
- Simplified core mechanic (shooting, jumping, collision, or collectible, depending on the chosen clone).
- Score and lives system.
- Short, repetitive gameplay loop, ideal for quick matches.

Genre

Arcade / Mini-arcade

Typical features: quick matches, increasing challenge, scoring, limited lives, simple mechanics.

Difficulty

3/10 (depending on the complexity of the chosen clone)

Despite the limited time, the project is technically simple: it applies concepts of input, loops, and basic AI, but requires quick decision-making, streamlined game design, and focus on functional gameplay.

Game Modes

- Solo: single player competes for the highest score.

Optional Extensions:

- Local versus: two players simultaneously in split-screen or shared arena.
- Local co-op: two players collaborating to survive longer.
- Online leaderboard: record fast scores.

Minimum Deliverables

To consider the project completed within the 48-hour timeframe, the following elements must be implemented:

Player Input and Movement

- Direct response to keyboard or joystick.
- Stable and predictable movement.

Core Mechanic of the Clone

- Attack, collection, or dodge, depending on the arcade type chosen.
- Minimal but functional interaction with enemies or obstacles.

Score System

- Correctly increment points for relevant actions (e.g., defeating an enemy, collecting an item).

Lives and Game Over

- Control of 3 lives (or a defined number) and proper game-ending when lives run out.

Functional Gameplay Loop

- The game must allow multiple quick attempts without crashing.

Optional Extensions

- More complex enemy AI (random movement or simple chasing).
- Temporary power-ups (speed, shield, double shot).
- Basic animations and sound effects.
- Level variations or increasing difficulty over time.
- Saving and displaying local high scores.

Learning Objectives

This project aims to teach and reinforce the following skills:

Rapid Gameplay Implementation

- Create a functional game under time pressure, reinforcing development efficiency.

Player Input and Game Loops

- Master response to direct controls and design short, repetitive gameplay loops.

Basic AI and Interaction

- Simple enemies or obstacles with predictable behavior.

Score and Lives Management

- Correctly increment score and handle functional game-over conditions.

Design Decision-Making Under Pressure

- Choose the clone, main mechanic, quick balancing, and minimum viable functionality.

This project is ideal for consolidating the ability to produce functional games in short periods, exercising rapid creation, prioritization of essential mechanics, and refinement of arcade gameplay.

BY GENRE — Inspired Projects

Organized in ascending order of difficulty. For each genre: 2 mandatory (core) projects and 3 optional ones. It is recommended to alternate with participation in Game Jams for quick and creative practice.

1. Simple Arcade (Medium Difficulty: 1–2/5)

Why: Games with direct input, short loops, and minimal AI, ideal for quickly learning fundamental mechanics.

Mandatory Projects

Expanded Pong (core expanded version)

- Mode: Local/Online Versus
- Minimum Deliverable: Functional online multiplayer
- Extensions: Tournaments and leaderboards

Expanded Space Invaders (core expanded version)

- Mode: Solo
- Minimum Deliverable: Infinite waves
- Extensions: Rotating power-ups

Optional Projects

Galaga

- Mode: Solo
- Minimum Deliverable: Double enemy formations
- Extensions: Multiple enemy capture

Expanded Asteroids

- Mode: Solo / Local Co-op
- Minimum Deliverable: Split-screen for co-op
- Extensions: Hyperspace in co-op

Centipede

- Mode: Solo
- Minimum Deliverable: Persistent mushrooms
- Extensions: Cooperative multiplayer

Game Jam: Endless Rush Jam (48h)

- Theme: 2D infinite runner
- Specifications: Procedurally generated obstacles, progressive difficulty, and power-ups

2. Clicker / Idle (Medium Difficulty: 2/5)

Why: Focused on incremental logic, economic systems, and save mechanics, essential for understanding game progression.

Mandatory Projects

Cookie Clicker

- Mode: Solo
- Minimum Deliverable: Basic clicks + 3 upgrades; functional save/load system
- Extensions: Prestige system

AdVenture Capitalist

- Mode: Solo
- Minimum Deliverable: Multiple businesses with revenue multipliers
- Extensions: Offline progression

Optional Projects

Clicker Heroes

- Mode: Solo
- Minimum Deliverable: Heroes that auto-click; ascension system
- Extensions: Boss encounters

Realm Grinder

- Mode: Solo
- Minimum Deliverable: Playable factions; reincarnations
- Extensions: Custom builds

Idle Miner Tycoon

- Mode: Solo
- Minimum Deliverable: Layered mines and automated managers
- Extensions: Seasonal events

Game Jam: Idle Loop Jam (48–72h)

- Theme: Prototype with 3 upgrades and prestige
- Specifications: Functional save/load system

3. 2D Static Puzzle (Medium Difficulty: 2/5)

Why: Games based on discrete rules, with little or no physics, perfect for developing logical thinking and level design.

Mandatory Projects

Tetris Expanded (expanded core version)

- Mode: Solo
- Minimum Deliverable: Marathon mode
- Extensions: Versus mode with garbage lines

Minesweeper

- Mode: Solo
- Minimum Deliverable: 10x10 grid, flags, victory by full clearance
- Extensions: Difficulty variations

Optional Projects

Sokoban

- Mode: Solo
- Minimum Deliverable: 10 levels, undo system
- Extensions: Customizable level editor

Bejeweled

- Mode: Solo
- Minimum Deliverable: 3+ matches and combos
- Extensions: Timed modes

Picross / Nonogram

- Mode: Solo
- Minimum Deliverable: 5x5 grids with hints
- Extensions: Various visual themes

Game Jam: Puzzle Sprint (48h)

- Theme: Grid-based puzzle with 10 levels
- Specifications: Hints, undo, 3-star system per level

4. 2D Infinite Runner (Difficulty: Medium 2/5)

Why: Light procedural generation, focus on pacing and rhythm.

Mandatory Projects

Flappy Bird Inspired (core expansion)

- Mode: Solo
- Minimum Deliverable: Power-ups
- Extensions: Skins

Canabalt Inspired

- Mode: Solo
- Minimum Deliverable: Auto-scrolling, double jumps
- Extensions: Procedurally generated buildings

Optional Projects

Chrome Dino Inspired

- Mode: Solo
- Minimum Deliverable: Random cacti, day/night cycle
- Extensions: Pterodactyl obstacles

Jetpack Joyride Inspired

- Mode: Solo
- Minimum Deliverable: Jetpack fuel, basic gadgets
- Extensions: Daily missions

Temple Run (2D) Inspired

- Mode: Solo
- Minimum Deliverable: Swipe to turn, power-ups
- Extensions: Variant maps

5. Visual Novel / Narrative (Difficulty: Medium 2/5)

Why: Focused on flag management, dialogue flow, and branching narratives—ideal for practicing interactive storytelling.

Mandatory Projects

Phoenix Wright Inspired

- Mode: Solo

- Minimum Deliverable: 3 cases with cross-examination
- Extensions: Evidence system and decision branches

Doki Doki Literature Club Inspired

- Mode: Solo
- Minimum Deliverable: Meta dialogues and 2 distinct endings
- Extensions: Customizable poems

Optional Projects

Steins;Gate Inspired

- Mode: Solo
- Minimum Deliverable: Messaging system, timelines
- Extensions: Multiple save slots

Katawa Shoujo Inspired

- Mode: Solo
- Minimum Deliverable: Romantic routes and minigames
- Extensions: Multiple possible endings

Long Live the Queen Inspired

- Mode: Solo
- Minimum Deliverable: Stats and random events
- Extensions: Different character classes

Game Jam: VN Weekend (72h)

- Theme: Short story with 3 endings
- Specifications: Flag system, decision inventory, and dialogue UI

6. Basic 2D Tower Defense (Dificuldade Média: 3/5)

Por quê: Focado em gerenciamento de caminhos, ondas de inimigos e upgrades de unidades, ideal para praticar design de sistemas estratégicos.

- **Projetos Obrigatórios**
    1. **Plants vs. Zombies Inspired**
        - **Modo:** Solo
        - **Entrega mínima:** 5 tipos de plantas, 10 waves
        - **Extensões:** Zen garden e personalização de plantas
    2. **Bloons TD Inspired**
        - **Modo:** Solo
        - **Entrega mínima:** 4 torres, paths curvos

        - **Extensões:** Upgrades de macacos
- **Projetos** **Opcionais**
    3. **Kingdom Rush Inspired**
        - **Modo:** Solo
        - **Entrega mínima:** Heróis e barreiras de defesa
        - **Extensões:** Torres especiais e habilidades únicas
    4. **GemCraft Inspired**
        - **Modo:** Solo
        - **Entrega mínima:** Criação de gems, labirintos (mazes)
        - **Extensões:** Orbs especiais
    5. **Defense Grid Inspired**
        - **Modo:** Solo / Co-op local
        - **Entrega mínima:** Torres compartilhadas em co-op
        - **Extensões:** Comando de voz do jogador

Game Jam: Tower Tactics Jam (48–72h)

- **Tema:** Mapa com 3 caminhos, 4 torres e 10 waves
- **Especificações:** Sistema de upgrades e progressão de dificuldade

7. Simple 2D Platformer (Difficulty: Medium 3/5)

Why: Focused on polished input, basic physics, and collisions—essential for understanding responsive gameplay.

Mandatory Projects

Super Mario Bros Inspired

- Mode: Solo
- Minimum Deliverable: 4 worlds, basic power-ups
- Extensions: Warp zones and hidden secrets

Classic Mega Man Inspired

- Mode: Solo
- Minimum Deliverable: 4 enemy robots, charge shot
- Extensions: E-tanks and special abilities

Optional Projects

Lode Runner Inspired

- Mode: Solo
- Minimum Deliverable: Digging system, guards with basic AI

- Extensions: Customizable level editor

Alex Kidd Inspired

- Mode: Solo
- Minimum Deliverable: Sections with vehicles
- Extensions: Interactive minigames

DuckTales Inspired

- Mode: Solo
- Minimum Deliverable: Pogo jump, treasure collection
- Extensions: Bosses and additional secrets

Game Jam: Platformer Sprint (48h)

- Theme: 4 screens + 1 boss
- Specifications: Functional checkpoints and progressive difficulty

8. Top-Down Shooter / Twin-Stick 2D (Difficulty: Medium 3/5)

Why: Focused on projectiles, enemy spawning, and performance optimization—ideal for practicing fast combat and action mechanics.

Mandatory Projects

Geometry Wars Inspired

- Mode: Solo / Local Co-op
- Minimum Deliverable: 4 weapons, combat arenas
- Extensions: Bombs and special power-ups

Hotline Miami Inspired

- Mode: Solo
- Minimum Deliverable: Top-down melee combat, masks
- Extensions: Procedural levels

Optional Projects

Nuclear Throne Inspired

- Mode: Solo
- Minimum Deliverable: Mutations, weapon variety
- Extensions: Co-op mode

Enter the Gungeon Inspired

- Mode: Solo
- Minimum Deliverable: Procedural rooms, dodge roll
- Extensions: Weapon and ability synergies

Smash TV Inspired

- Mode: Local Co-op
- Minimum Deliverable: Arena waves, power-ups
- Extensions: Host variations

Game Jam: Twin-Stick Jam (48h)

- Theme: Arena with 3 enemy types, 4 weapons, dynamic spawning, and scoring system
- Specifications: Smooth combat mechanics and difficulty balancing

9. 2D Beat'em Up / Hack-n-Slash (Difficulty: Medium 3/5)

Why: Focused on melee collisions, combos, and fluid combat mechanics—ideal for studying rhythm and control responsiveness.

Mandatory Projects

Streets of Rage 2 Inspired

- Mode: Solo / Local Co-op
- Minimum Deliverable: 3 stages, basic combos
- Extensions: Special abilities

Final Fight Inspired

- Mode: Local Co-op
- Minimum Deliverable: Grab throws and bosses
- Extensions: Interactive vehicles

Optional Projects

Double Dragon Inspired

- Mode: Local Co-op
- Minimum Deliverable: Weapon pickups
- Extensions: Advanced combat system

River City Ransom Inspired

- Mode: Solo
- Minimum Deliverable: RPG stats and shops
- Extensions: Schools and interactive locations

Castle Crashers Inspired

- Mode: Online Co-op
- Minimum Deliverable: Animal orbs
- Extensions: Levels with magic

Game Jam: Beat'em Up Jam (72h)

- Theme: 1 stage, 3 enemies, 2 combos, optional local co-op
- Specifications: Responsive combat mechanics and simple progression

10. 2D Turn-Based RPG (Difficulty: Medium 4/5)

Why: Focused on managing stats, inventory, and combat balancing—ideal for studying complex progression systems and strategic design.

Mandatory Projects

Pokémon Red/Blue Inspired

- Mode: Solo
- Minimum Deliverable: 1 starter Pokémon, 5 routes, basic battles
- Extensions: Gyms

Final Fantasy (NES) Inspired

- Mode: Solo
- Minimum Deliverable: Party of 4 characters, spells
- Extensions: Crystals and story mechanics

Optional Projects

Dragon Quest Inspired

- Mode: Solo
- Minimum Deliverable: Character classes, inns
- Extensions: Vehicles and transportation

Undertale Inspired

- Mode: Solo
- Minimum Deliverable: Mercy system, branching dialogues
- Extensions: Genocide route

Secret of Mana Inspired

- Mode: Local Co-op
- Minimum Deliverable: Ring commands for abilities
- Extensions: Spirits and special skills

Game Jam: JRPG Mini-Jam (72h)

- Theme: 1 village, 1 dungeon, turn-based combat, XP system
- Specifications: Progression mechanics, enemy and item balancing

11. 2D Metroidvania (Difficulty: Medium 4/5)

Why: Focused on interconnected maps and gating—ideal for practicing exploration, skill progression, and non-linear level design.

Mandatory Projects

Metroid Inspired

- Mode: Solo
- Minimum Deliverable: 10 rooms, Morph Ball
- Extensions: Missiles and additional power-ups

Castlevania: Symphony of the Night Inspired

- Mode: Solo
- Minimum Deliverable: Sub-weapons, castle map
- Extensions: Souls system and upgrades

Optional Projects

Axiom Verge Inspired

- Mode: Solo
- Minimum Deliverable: Glitch Gun, basic hacks
- Extensions: Drones and advanced abilities

Hollow Knight Inspired

- Mode: Solo
- Minimum Deliverable: Nail Arts, charms
- Extensions: Pantheons and extra challenges

Ori (2D) Inspired

- Mode: Solo
- Minimum Deliverable: Dash, Spirit Wells
- Extensions: Skill tree and upgrades

Game Jam: Metroidvania Lite (7 days)

- Theme: 12 rooms, 2 abilities that unlock paths, 1 mini-boss
- Specifications: Skill progression, non-linear exploration, and strategic checkpoints

12. 2D Roguelike / Roguelite (Difficulty: Medium 4/5)

Why: Focused on procedural generation and permadeath—ideal for studying dynamic systems, progression, and randomly designed challenges.

Mandatory Projects

Spelunky Inspired

- Mode: Solo
- Minimum Deliverable: Procedurally generated caves, ropes
- Extensions: Ghosts and random events

The Binding of Isaac Inspired

- Mode: Solo
- Minimum Deliverable: Procedural rooms, basic items
- Extensions: Item synergies and power-ups

Optional Projects

Rogue Inspired

- Mode: Solo
- Minimum Deliverable: ASCII-style grid, permadeath
- Extensions: Potions and special effects

NetHack Inspired

- Mode: Solo
- Minimum Deliverable: Complex items, pets
- Extensions: Classes (Roles) and unique abilities

Dead Cells Inspired

- Mode: Solo
- Minimum Deliverable: Run progression, weapons
- Extensions: Boss Cells and advanced challenges

Game Jam: Rogue Jam (7 days)

- Theme: Procedural dungeon, basic loot, permadeath
- Specifications: Progression mechanics, challenge balancing, and consistent procedural generation

13. 2D Farming / Crafting / Survival (Difficulty: Medium 4/5)

Why: Focused on world cycles, inventory management, and progression—ideal for studying complex survival and crafting mechanics.

Mandatory Projects

Stardew Valley Inspired

- Mode: Solo / Online Co-op (up to 4 players)
- Minimum Deliverable: Fields and irrigation, seeds, cave, NPCs with daily routines, day/night cycle
- Extensions: Seasonal events, animals, marriage

Don’t Starve Inspired (2D Demake)

- Mode: Solo / Online Co-op
- Minimum Deliverable: Hunger, health, and sanity bars; campfire; biomes; essential crafting
- Extensions: Bosses, caves, and perks

Optional Projects

Graveyard Keeper Inspired

- Mode: Solo
- Minimum Deliverable: Quality graves, workshops, tech tree
- Extensions: Sermons, trading, and automation

Sun Haven Inspired (2D Demake)

- Mode: Solo / Online Co-op
- Minimum Deliverable: Farm combined with combat, skills, quests
- Extensions: Talent trees, dungeons, and pets

Starbound Inspired

- Mode: Solo / Online Co-op
- Minimum Deliverable: Varied planets, mining and crafting, simple quests
- Extensions: Spaceships, colonies, and events

Game Jam: Survive & Craft Jam (14 days)

- Theme: Small island, 10 crafting recipes, hunger and energy mechanics
- Specifications: Day/night cycle, crafting progression, and resource management

14. 2D City-Builder (Dificuldade Média: 4/5)

Por quê: Focado em gestão urbana e simulação, ideal para estudar planejamento, recursos e sistemas interdependentes.

- **Projetos Obrigatórios**
    1. **SimCity Inspired (2D Demake)**
        - **Modo:** Solo
        - **Entrega mínima:** Malha viária, zonas residenciais/comerciais/industriais, impostos, demanda e desastres
        - **Extensões:** Transporte público, políticas urbanas
    2. **Cities: Skylines Inspired (2D Demake)**
        - **Modo:** Solo
        - **Entrega mínima:** Pathfinding de tráfego, serviços básicos, overlays de informação

        - **Extensões:** Distritos, mods de regras
- **Projetos Opcionais**

3. **Islanders Inspired**
    - **Modo:** Solo
    - **Entrega mínima:** Deck de peças, pontuação por adjacência, ilhas sequenciais
    - **Extensões:** Biomas variados, peças raras, desafios diários
4. **Banished Inspired**
        - **Modo:** Solo
        - **Entrega mínima:** Ciclos de comida e madeira, estações do ano, profissões básicas
        - **Extensões:** Doenças, comércio e eventos aleatórios
5. **Anno Inspired (2D Demake)**
        - **Modo:** Solo
        - **Entrega mínima:** Rotas de comércio, níveis de residentes, cadeias de produção 3–4 passos
        - **Extensões:** Diplomacia, IA concorrente e eventos estratégicos

15. 2D Tycoon / Economy (Difficulty: Medium 4/5)

Why: Focused on logistics, finance, and resource management—ideal for practicing strategic planning and complex economic systems.

Mandatory Projects

RollerCoaster Tycoon Inspired (2D Demake)

- Mode: Solo
- Minimum Deliverable: Attractions, editable tracks, visitors, loan management
- Extensions: Campaigns, marketing, and special events

Transport Tycoon / OpenTTD Inspired

- Mode: Solo
- Minimum Deliverable: Industry demands, stations, profitable routes
- Extensions: Advanced signals, competitor AI

Optional Projects

Game Dev Tycoon Inspired

- Mode: Solo
- Minimum Deliverable: Game themes and genres, reviews, team management
- Extensions: Own consoles, MMO development

Prison Architect Inspired (2D Focus)

- Mode: Solo
- Minimum Deliverable: Cells, prison regimes, finances
- Extensions: Rehabilitation programs, riots

Anno Inspired (City-Builder Expansion)

- Mode: Solo
- Minimum Deliverable: Specialized islands, naval trade
- Extensions: Expeditions and strategic events

Post-Advanced Genres Game Jam: Open-World Mini Jam (28 days)

- Theme: Tile-based world with chunks
- Specifications: 3 interactive NPCs, 2 dynamic quests, basic gathering, and initial resource progression

---

Additional Game Jams (Provocative Collection – 20 Escalated Ideas)

Objective: Creative exercises between study projects. Each Jam includes: theme, constraint, mechanical twist, game mode, duration, minimum deliverable, and possible optional extensions. Difficulty rated 1–5/5.

Single Pulse – Difficulty 1/5 | Solo | 2h

- **Theme:** World controlled by a single button
- **Constraint:** Single input
- **Twist:** Button charges harmony, generating musical curves
- **Minimum Deliverable:** Functional trial-and-error loop + 3 music tracks
- **Extensions:** Daily seeds for replayability

Sliding Synapses – 1/5 | Solo | 2h

- **Theme:** Connect points to create paths
- **Constraint:** No explanatory text
- **Twist:** Each connection redefines level rules
- **Minimum Deliverable:** 15 short playable levels
- **Extensions:** Level editor for custom content

RhythmoGraph – 1/5 | Solo | 2h

- **Theme:** Draw rhythm visually

- **Constraint:** No HUD or explicit indicators
- **Twist:** Player strokes transform into rhythmic cues
- **Minimum Deliverable:** 3 songs + 2 rhythm brushes
- **Extensions:** Composer mode for custom music creation

Micro-Platforms – 1/5 | Solo | 8h

- **Theme:** Micro-levels in 10x6 grids
- **Twist:** Death reduces gravity for 10 seconds
- **Minimum Deliverable:** 24 complete rooms + dash implemented
- **Extensions:** Ghost runs to compare attempts

Orbital Duet – 2/5 | Solo / Local Co-op | 8h

- **Theme:** Two bodies orbiting without scrolling
- **Twist:** Shots alter gravity
- **Minimum Deliverable:** 1 functional arena + enemy waves
- **Extensions:** Levels with a boss

Word Echo – 2/5 | Solo | 8h

- **Theme:** Minimalist narrative choices
- **Twist:** Each decision echoes across three future chapters
- **Minimum Deliverable:** 5 chapters + decision journal
- **Extensions:** Visual map of routes and consequences

Micro-Fest – 2/5 | Local Versus | 24h

- **Theme:** Short minigames (3–7 seconds)
- **Twist:** Rules dynamically change via cards
- **Minimum Deliverable:** 5 minigames + round system
- **Extensions:** Local leaderboards

Living Maze – 2/5 | Solo | 24h

- **Theme:** Walls grow organically
- **Twist:** Player prunes walls to create paths
- **Minimum Deliverable:** 10 levels + pruning tool
- **Extensions:** Mutant seeds to generate varied mazes

Shadow Echo – 3/5 | Local Co-op | 48h

- **Theme:** Partner’s shadow acts as an enemy
- **Twist:** Lights must be synchronized
- **Minimum Deliverable:** 5 rooms + light puzzles
- **Extensions:** Solo mirror mode for individual practice

Reversed Time – 3/5 | Solo | 48h

- **Theme:** Local time manipulation
- **Twist:** Cumulative effects impact gameplay
- **Minimum Deliverable:** 8 puzzles + rewind button
- **Extensions:** Branching timelines

Collective Hunger – 3/5 | Online Versus | 48h

- **Theme:** Resources shared among players
- **Twist:** Map shrinks over time
- **Minimum Deliverable:** 2v2 arena + basic loot
- **Extensions:** Alliance and betrayal system

Overlapping Dreams – 3/5 | Solo / Online Co-op | 72h

- **Theme:** Dream layers alter physics
- **Twist:** Seamless transitions between layers
- **Minimum Deliverable:** 3 layers + layer switches
- **Extensions:** Dream editor for custom levels

Black Market – 4/5 | Solo | 72h

- **Theme:** Procedural economy with cheating AI
- **Twist:** Rumors and random events affect prices
- **Minimum Deliverable:** 5 items + functional trading system
- **Extensions:** Black market events for variability

Shared Body – 4/5 | Local Co-op | 72h

- **Theme:** One body controlled by two minds
- **Twist:** Input conflicts cause gameplay chaos
- **Minimum Deliverable:** 4 levels + merge UI
- **Extensions:** Split control modes

Fragmented Memory – 4/5 | Solo | 7 days

- **Theme:** Puzzles reconstruct memories
- **Twist:** False fragments confuse the player
- **Minimum Deliverable:** 10 shards + logical verification
- **Extensions:** Multiple branched “truths”

Pixel Rebellion – 4/5 | Local/Online Versus | 7 days

- **Theme:** Pixels rebel on screen
- **Twist:** Player must fix via direct input
- **Minimum Deliverable:** Pixel waves + basic tools

- **Extensions:** Custom rebellion system

Eternal Cycle – 5/5 | Solo / Online Co-op | 14 days

- **Theme:** Life/death loop affects persistent world
- **Twist:** Legacies are inherited between cycles
- **Minimum Deliverable:** 3 full cycles + skill inheritance
- **Extensions:** Multi-generational replay

Conspiring Shadows – 5/5 | Solo | 14 days

- **Theme:** Shadows conspire against light
- **Twist:** Eavesdropping on shadow dialogues alters decisions
- **Minimum Deliverable:** Branching narrative + stealth gameplay
- **Extensions:** Shadow alliances for emergent interactions

Quantum Flow – 5/5 | Online Co-op | 21 days

- **Theme:** Probabilistic quantum states
- **Twist:** Player observation collapses realities
- **Minimum Deliverable:** 5 quantum puzzles + player synchronization
- **Extensions:** Parallel universes for replay and exploration

Infinite Echo – 5/5 | Online Versus | 21 days

- **Theme:** One player's actions echo in the opponent's world
- **Twist:** Chain reactions based on karma
- **Minimum Deliverable:** 1v1 arenas + echo system
- **Extensions:** Tournament mode with global ranking

Final Tips:

- Keep a detailed progress journal for each Jam.
- Prioritize polish and game feel after the minimum deliverable.
- For multiplayer, implement simple P2P first.
- Share in jams to receive external feedback.
- Focus on completing at least the minimum deliverable before adding extensions.