SpaceScrapers - Game Documentation

Table of Contents

Game Overview

SpaceScrapers is a physics-based tower building game where players construct tall structures using limited tile resources while facing environmental challenges across different planetary environments.

Core Concept

- **Objective**: Build towers that reach target heights and survive environmental challenges
- **Gameplay**: Strategic tile placement followed by realistic physics simulation
- **Challenge**: Limited resources and environmental hazards test your engineering skills
- **Theme**: Space exploration across Earth, Jupiter, and Mars environments

Game Flow

1. **Building Phase**: Drag and drop tiles to construct your tower
2. **Simulation Phase**: Watch your tower face physics and environmental challenges
3. **Results**: Success depends on reaching target height AND surviving challenges

---

Installation & Requirements

System Requirements

- **Python**: 3.7 or higher
- **Operating System**: Windows, macOS, or Linux
- **Memory**: 512 MB RAM minimum
- **Graphics**: Basic graphics support (OpenGL recommended)

Installation Steps

1. **Install Python Dependencies**

```
pip install pygame numpy
```

2. **Download Game Files**
    - Extract SpaceScrapers folder to desired location
    - Ensure all assets are in the `assets/` directory
3. **Run the Game**

```
cd SpaceScrapers
python main.py
```

External Libraries

- **Pygame**: Graphics, input handling, and game loop
- **NumPy**: Physics calculations and vector mathematics

How to Play

Getting Started

1. **Launch Game**: Run `python main.py`
2. **Watch Intro**: Enjoy the SpaceScrapers introduction
3. **Main Menu**: Click "Play" to access level selection
4. **Choose Level**: Select from Earth, Jupiter, or Mars environments

Building Phase

1. **Select Tiles**: Available tiles appear in the bottom panel
2. **Drag & Drop**: Click and drag tiles from selection area to build area
3. **Placement Rules**:

    - Tiles cannot overlap (pixel-perfect collision detection)
    - Invalid placements return tiles to selection area
    - No rotation available - focus on strategic placement
4. **Start Simulation**: Press SPACE when ready to test your tower

Simulation Phase

1. **Physics Activation**: All tiles become subject to gravity and physics
2. **Environmental Challenges**: Level-specific hazards activate:
    - **Earth**: Wrecking ball swings across the screen
    - **Jupiter**: Strong winds push tiles horizontally
    - **Mars**: Meteorites rain down from space
3. **Survival Timer**: Endure 12 seconds of challenges
4. **Height Check**: Tower must reach target line AND survive

Win/Lose Conditions

Victory Requirements:

- Tower reaches the target height line
- Structure survives the full 12-second challenge period
- At least part of the tower remains stable

Defeat Conditions:

- Tower collapses completely
- Insufficient height reached
- All tiles fall below target line

Game Features

Advanced Physics System

Realistic Mechanics:

- Gravity simulation with configurable strength
- Friction and damping for natural movement
- Collision response with bounce and energy loss
- Angular momentum and tumbling effects
- Support detection for structural stability

Visual Feedback:

- Tiles rotate visually when becoming unstable
- Particle effects for impacts and collisions

- Smooth animations separate from physics calculations
- Color-coded progress indicators

Tile Types & Properties

Rectangle Tiles:

- Versatile building blocks
- Good for foundations and walls
- Balanced mass and friction properties

Square Tiles:

- Compact and stable
- Excellent for solid construction
- Higher mass provides stability

Beam Tiles:

- Long and narrow design
- Perfect for bridges and horizontal supports
- Lower mass but strategic shape

Environmental Challenges

Earth Level - Wrecking Ball:

- Heavy pendulum swings across build area
- Tests structural integrity through impact
- Timing and positioning critical for survival

Jupiter Level - Wind Effects:

- Horizontal forces push tiles sideways
- Tests foundation strength and balance
- Requires wide, stable base construction

Mars Level - Meteorite Shower:

- Projectiles fall from random positions
- Explosive impacts create debris
- Tests overall structural robustness

Visual Effects System

Particle Systems:

- Explosion effects for meteorite impacts

- Dust clouds for collision feedback
- Sparkle effects for successful tile placement
- Wind particles showing force direction

Animation Features:

- Smooth tile rotation for instability feedback
- Pulsing target line for clear objectives
- Fade effects for temporary UI elements
- Impact flashes for collision highlights

Level Guide

Level 1: Earth - "First Steps"

Environment: Peaceful Earth landscape **Target Height**: 400 pixels **Available Tiles**: 2 Rectangles, 2 Beams, 2 Squares **Challenge**: Wrecking Ball

Strategy Tips:

- Build a stable foundation using squares
- Use beams for horizontal reinforcement
- Position structure away from wrecking ball path
- Focus on low, stable construction rather than pure height

Challenge Details:

- Wrecking ball appears after 2 seconds
- Swings from right to left across screen
- Heavy impact force requires solid foundations

Level 2: Jupiter - "Storm Winds"

Environment: Turbulent Jupiter atmosphere **Target Height**: 500 pixels **Available Tiles**: 3 Rectangles, 3 Beams, 1 Square **Challenge**: Wind Forces

Strategy Tips:

- Create a wide base to resist wind
- Use interlocking tile patterns
- Build progressively narrower towards top
- Position heaviest tiles at bottom

Challenge Details:

- Horizontal wind forces activate early

- Consistent lateral pressure throughout simulation
- Tests foundation width and structural balance

Level 3: Mars - "Meteor Storm"

Environment: Rocky Martian landscape **Target Height**: 650 pixels **Available Tiles**: 2 Squares, 3 Rectangles, 4 Beams
Challenge: Meteorite Impacts

Strategy Tips:

- Build tall, thin structures to minimize target area
- Use redundant support columns
- Create flexible joints using beam placement
- Plan for partial collapse and recovery

Challenge Details:

- Random meteorite impacts throughout simulation
- Explosive force creates debris and shockwaves
- Multiple impact points test structural redundancy

Technical Architecture

Module Structure

```
SpaceScrapers/
├── main.py                 # Game orchestration and main loop
├── tile_placement.py       # Physics simulation and tile management
├── challenges.py           # Environmental challenge systems
├── animations.py           # Visual effects and particle systems
├── tiles.py               # Tile definitions and graphics
├── sprite_manager.py      # Asset loading and management
├── game/
│   └── level.py           # Level rendering and setup
├── screens/
│   ├── intro.py           # Introduction sequence
│   ├── menu.py            # Main menu interface
│   ├── level_select.py    # Level selection screen
│   ├── win.py             # Victory screen
│   └── lose.py            # Defeat screen
├── storage/
│   └── level_storage.py   # Level data and configuration
├── assets/                # Graphics and media files
└── sprites/               # Sprite sheets and tile graphics
```

Key Classes

Tile Class (`tile_placement.py`):

- Manages individual tile physics and rendering
- Handles collision detection and response
- Implements support detection algorithms
- Provides visual rotation feedback

TilePlacer Class (`tile_placement.py`):

- Orchestrates all tile interactions
- Manages drag-and-drop mechanics
- Runs physics simulation loops
- Validates tile placement rules

ChallengeManager Class (`challenges.py`):

- Coordinates level-specific environmental hazards
- Manages timing and intensity of challenges
- Handles challenge-tile interactions

AnimationManager Class (`animations.py`):

- Controls all visual effects and particles

- Manages animation timing and lifecycle
- Provides feedback for game events

Design Patterns

Separation of Concerns:

- Physics calculations separate from visual effects
- Game logic independent of rendering
- Asset management abstracted from gameplay

Component-Based Architecture:

- Modular challenge system
- Pluggable animation effects
- Extensible tile type system

Physics System

Core Mechanics

Gravity Simulation:

- Constant downward acceleration (500 pixels/sec²)
- Selective gravity based on support detection
- Smooth physics integration using delta time

Collision Detection:

- Pixel-perfect collision using Pygame masks
- Separate collision for placement vs. physics
- Efficient bounding box pre-filtering

Support Detection:

- Three-point bottom-edge detection system
- Center, left corner, and right corner sampling
- Smart tumbling only when inadequately supported

Physics Properties

Per-Tile Properties:

- **Mass**: Affects momentum and impact force
- **Friction**: Controls velocity damping (0.98 default)
- **Restitution**: Bounce factor for collisions (0.2 default)

- **Inertia**: Rotational resistance

Global Constants:

- **Gravity**: 500 pixels/second²
- **Max Velocity**: 300 pixels/second (capped for stability)
- **Static Threshold**: 10 pixels/second (tiles become static)

Advanced Features

Tumbling Physics:

- Realistic angular momentum calculation
- Visual rotation feedback (±45° maximum)
- Smart tumbling only when tiles are airborne
- Stability detection for well-supported tiles

Collision Response:

- Impulse-based collision resolution
- Energy conservation with damping
- Separate normal and tangential forces
- Ground collision with bounce damping

Controls Reference

Mouse Controls

- **Left Click + Drag**: Move tiles from selection to build area
- **Mouse Hover**: Highlight interactive elements
- **Click**: Activate menu buttons and UI elements

Keyboard Controls

- **SPACE**: Start physics simulation (building phase only)
- **ESC**: Return to main menu (from any screen)
- **R**: Reset current level (building phase only)
- **Any Key**: Skip intro sequence or objective screen

UI Navigation

- **Level Selection**: Click on planet icons
- **Menu Navigation**: Click buttons or use mouse
- **Game Phases**: Automatic progression through building → simulation → results

---

Troubleshooting

Common Issues

Game Won't Start:

- Verify Python 3.7+ installation
- Check Pygame and NumPy are installed: `pip list`
- Ensure all game files are in same directory
- Run from command line to see error messages

Graphics Issues:

- Update graphics drivers
- Try running in compatibility mode
- Check for OpenGL support
- Verify sprite files exist in `sprites/` directory

Performance Problems:

- Close other applications
- Lower screen resolution if possible
- Check available RAM (minimum 512MB)
- Disable background processes

Physics Glitches:

- Ensure tiles aren't overlapping during placement
- Reset level if tiles get stuck
- Check for very high velocities causing instability

Error Messages

"AttributeError: 'Tile' object has no attribute 'is_static'"

- Indicates corrupted tile initialization
- Restart the game
- Check for modified game files

"No module named 'pygame'"

- Install Pygame: `pip install pygame`
- Verify Python environment
- Use virtual environment if conflicts exist

"Could not load spritesheet"

- Check `sprites/` folder exists
- Verify sprite files are present
- Ensure file permissions allow reading

Performance Optimization

For Slower Systems:

- Reduce particle density in `animations.py`
- Lower frame rate cap in main loop
- Disable debug visualizations
- Use smaller window size

Development Notes

Code Architecture Decisions

Physics vs. Visual Separation:

- Visual rotation is cosmetic only
- Physics collision always uses upright orientation
- Ensures predictable gameplay behavior

Pixel-Perfect Collision:

- Uses Pygame mask collision for accuracy
- More expensive than bounding box collision
- Essential for satisfying tower building mechanics

Static Tile Optimization:

- Tiles become static when velocity drops below threshold
- Prevents unnecessary physics calculations
- Improves performance with many tiles

Extensibility Features

Adding New Tile Types:

1. Create new class in `tiles.py` inheriting from `TileShape`
2. Define size, mass, friction, and restitution properties
3. Add sprite mapping in sprite manager
4. Update level configurations to include new tile type

Creating New Challenges:

1. Add new challenge class in `challenges.py`
2. Implement update(), draw(), and interaction methods
3. Register challenge in `ChallengeManager`
4. Configure in level data with timing parameters

Level Creation:

1. Add new level data in `level_storage.py`
2. Include background assets in `assets/` folder
3. Define tile allocation and target height
4. Specify challenge type and parameters

Technical Constraints

Library Limitations:

- Only Pygame and NumPy allowed
- No external physics engines
- Custom collision detection required

Performance Considerations:

- O(n²) collision detection between tiles
- Mask collision is computationally expensive
- Particle systems require careful memory management

Platform Compatibility:

- Cross-platform Python/Pygame support
- Asset loading works on Windows/Mac/Linux
- No platform-specific dependencies

Conclusion

SpaceScrapers demonstrates sophisticated game development using only Pygame and NumPy, featuring realistic physics simulation, engaging gameplay mechanics, and polished visual effects. The game successfully combines strategic puzzle elements with physics-based challenges to create an entertaining and educational experience.

The modular architecture ensures extensibility for future features while maintaining clean separation between game systems. The physics engine provides realistic and satisfying tile interactions that reward thoughtful construction strategies.

Whether you're building your first tower on Earth or surviving the meteor storms of Mars, SpaceScrapers offers a unique and challenging gaming experience that tests both engineering intuition and quick reflexes.

Game Version: 1.0

Documentation Version: 1.0

Last Updated: June 15, 2025

Author: Matúš Várfalvy