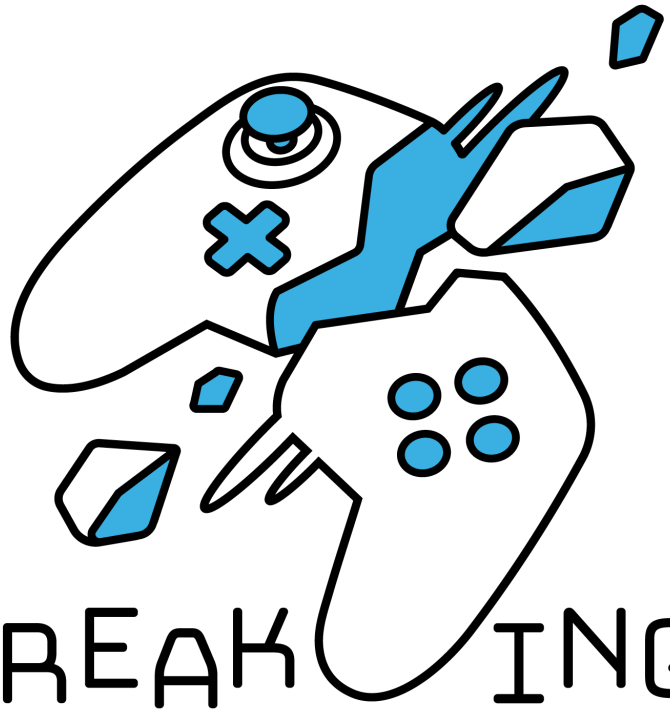

BREAKING CONVENTIONS

2022-2023

Achievement Guide

16th Annual Oregon Game Project Challenge

www.ogpc.info

OREGON GAME
PROJECT CHALLENGE

Overview

Achievements are broken down into multiple levels. To qualify for a higher level, a team must fully complete all requirements from the lower levels. Even if it would be possible to complete some part of a higher level without completing everything from the lower level, a team will only qualify for the lowest fully completed level. Teams will only qualify for *completed* levels – i.e. all requirements for the level have been met. For example, a team that has not fully completed the standard level cannot qualify for either encouraged nor extra credit levels.

The achievements are broken into the following levels:

- **Standard**: New teams should try to target the Standard level of every achievement. These are achievements we feel should be attainable by students of any skill level.
- **Encouraged**: The Encouraged level is slightly more advanced than the Standard level. These are achievements that most teams should work towards, even if you can't quite get all of them.
- **Extra Credit**: The Extra Credit level is the most advanced. Teams targeting this level will have to go above and beyond to consistently reach the Extra Credit level.

Achievements At a Glance

PROGRAMMING

Maintenance of Code

Code should be easy to follow, saved in a central location with backups, and have development over time. No spaghetti code!

Standard

- Code is clean: labeled properly and named descriptively
- Save multiple copies of the project, or project is saved to the cloud/online

Encouraged

- Code is navigable and logically separated into functional units
- Git/version control history shows at least a few commits over time

Extra Credit

- General “best practices” were followed and the code would allow for future development
- Git/version control history shows many commits, smaller or even daily commits

Explanation

This achievement focuses on your use of source control, or a method of backing up code, and cleanliness of code. These are both fundamental to maintaining a good codebase and allow others to easily step in and orient themselves with new code.

Source control is a critical part of keeping track of your work, how it looks over time, and knowing who made each change. Almost every professional development group uses it to keep track of their code. Solutions like Dropbox, Google Drive, and OneDrive are an acceptable form of version control starting point and ensure that your game will survive losing that one flash drive! For advanced teams, Git, Mercurial, or Azure DevOps are more professional grade source control solutions. There is a lot of documentation on these options so feel free to give them a try!

In addition to source control, code should be easy to read with simple names and easy to follow logic. This also includes comments, tabbing, semicolons, and functions that are as simple as possible (no mega functions that do everything!). Visual coding languages (such as Scratch) should be grouped into logical sections and be labeled. Example for textual code:

```
class Actor {
    constructor() {
        // Health will be overwritten when extending
        this.health = 1;
        this.alive = true;
    }
    // If health is too low, actor will be removed from scene
    takeDamage(damage) {
        this.health -= damage;
        if (this.health <= 0)
            this.alive = false;
    }
}
```

For the higher tiers, make sure your code is not only named appropriately, but also grouped appropriately. For source control, commit early and often! Try to avoid saving code you know is broken but try to commit as often as you possibly can. Frequent backups or small saves are much more helpful than infrequent ones where you risk losing lots of work.

Bag O' Tools

How did you leverage conditionals/repetition/data structures to implement a mechanic and diagram logic?

Standard

- Code has loops and conditionals
- Code uses at least 1 standard data structure (a list/map/dictionary is ok)

Encouraged

- Code uses loops and conditionals consistently and properly
- Code uses data structures in multiple locations
- Code comments include a high level description of at least one mechanic's implementation, preferably as pseudocode or a block diagram

Extra Credit

- Game implements and documents an algorithm
- Player state stored in a reasonable data structure (not a group of global variables)
- Codebase uses classes or similar data/functionality grouping structures where appropriate

- Code comments include a high level description of at least two mechanics' implementations,preferably as pseudocode or a block diagram

Explanation

Loops and conditionals are a basic part of keeping code a reasonable length and easy to read. As a codebase gets larger, the use of consistent paradigms makes it easier to find bugs quickly without needing to parse unfamiliar code. In addition to consistent code paradigms, explicit diagraming can also make it easier to conceptualize the code and find issues in advance.

Understanding the data flow, UI flow, activity flow, or states your game has is important for keeping track of your game as it grows. They can also be used to help bring another person onto your team without needing to verbally describe everything. UML (Unified Modeling Language) is a good place to start with this one as it contains well documented descriptions of many types of charts and diagrams you can make for your game.

For higher tiers, make sure to use loops, conditionals, and data structures frequently and consistently. Diagram more difficult sections of code (like your algorithm for example) and avoid global variables (they're evil).

Bug Smashing

Show how problems were fixed, avoided, and purged!

Standard

- Talk about or show a logic bug (not typo) and talk about or show how it was fixed
- Game demonstration doesn't crash, freeze, or otherwise completely break

Encouraged

- Talk about a proactive strategy you used to avoid bugs
- Gameplay demonstration is free of major/obvious bugs
- Have a testing process to help search for bugs

Extra Credit

- Have a system to keep track of bugs you've noticed (a list works fine)
- Game executes flawlessly during demonstration
- Document how your team leveraged a standard testing framework

Explanation

A bug describes a mistake in the logic of your code. It's more than just a typo. If you forget a semicolon and your code stops compiling, just add a semicolon and move on. A logic bug is specifically when you expected a variable to be in a certain state, and it

wasn't. These bugs often require the programmer to make changes to their code: adding a branch of logic to handle a specific case, making sure your math is correct, or even just making sure that you're not overstepping the end of an array. The changes required to fix a logic bug are usually more complicated than a simple typo.

A well-built game should perform well, even when being watched! If the demonstrator messes up and doesn't do something right, that is not a bug. Glitching through walls, breaking the physics engine, failing to pick up an item on keypress, or causing the game to stop responding or even crash are all to be avoided!

Comprehensive testing strategies help prevent bugs from reaching the player and ruining the game's experience. You should test your game (pretty please). Testing helps to locate bugs and ensure everything is working as intended. In the industry, testing frameworks come in a variety of flavors such as Jasmine (many languages), Crispy (GameMaker), Pytest (Python), or Jest (JavaScript) depending on what language you use. These frameworks add another tool to your bug smashing toolbelt!

For higher tiers, we're looking for clearly defined systems for finding, tracking, and smashing bugs. These systems will (hopefully) help produce a cleaner product that will run smoother when demoing for the judges.

Dynamic Aspects

Games make use of time and other modular elements to enhance gameplay.

Standard

- Game features more than one version of at least one asset to represent different states and dynamically swaps those assets at least once (for example, player can pick up and wear a yellow shirt)
- Animate something with a state machine (for example, swap 2 walking sprites)

Encouraged

- Game features a time related system which meaningfully affects gameplay (for example, day/night, weather cycles, etc.)
- Game swaps/combines/modifies assets or modifies mechanics based on time system (for example, day/night cycles, seasons, in one level; music speeding up as the countdown nears zero; etc.)

Extra Credit

- Game dynamically swaps or combines assets to match player actions
- Choose at least one of these three:
    - Game features a full-featured time system integrated into major mechanics of the game

    - Game generates content procedurally
    - Game has dynamic or procedural level generation

Explanation

Introducing an in-game day/night cycle, weather pattern, or other similar time-based event system is one way to vary gameplay, introduce some interesting programming challenges, and encourage advanced planning. Many popular games make use of rain, sunlight, sandstorms, snow, fog, and other such atmospheric cycles. Think of games such as Stardew Valley with its weather and day/night cycles, Pokémon games such as Sword and Shield with berries respawning after 24 hours or the day/night cycle, or Pokémon Go with the various Eevee evolutions. While it would be tempting to simply swap assets for darker ones on a given map, we're looking for a real-time change (since this is a programming achievement!). Think about how a level or map could change, what events could be introduced, or which items can be picked up depending on in-game time passing.

In addition to the various time cycles swapping assets, for the highest tier your game should swap assets based on player actions. This could be layering music as the player enters a spooky cave, a lever with multiple states, or the player's nose growing longer every time they tell a lie.

In addition to swapping assets the highest tier also asks for procedural content or level generation. This can be achieved in several different ways. Minecraft is a great example of procedural content generation in level design: clouds can be generated procedurally, shrubbery or other plants could be generated, music could be generated, textures could be generated, the map could be generated, and pretty much anything else in the game.

The big thing is, something has to come out of the generator that is unique. A system such as random player stats does enhance replayability, but is not procedural *content* or *level* generation.

Object Permanence

Menus allow players to interact with additional game content and access saved data.

Standard

- Game has at least one menu with an option to view credits
- Game has a leaderboard in memory (it does not need to persist between launches) for at least one statistic (for example: score, time to complete story, number of clicks, etc.)

Encouraged

- Game has a start menu with options to:
    - Start a new game

    - Use saved data (view the leaderboard, resume a saved game, or see other saved statistics)
    - View credits
    - Change options/settings (at least 2 functional, such as volume, difficulty, or keybindings)
- Game has at least one persistent *local* record such as a save system, leaderboard, settings, or other game related data

Extra Credit

- Game has a pause menu with resume, and at least 4 functional game settings (volume, difficulty, keybindings, etc)
- Game has a persistent *online* record such as a save system, leaderboard, or other game related data (server doesn't need to be hosted)

Explanation

Menus provide a buffer to your players between launching and playing the game. They also allow the player to pause and take a bathroom break, adjust settings, or access leaderboards or other saved data. Menus can be in game, a separate scene, their own controls, or whatever works in your engine, just make sure all menu options are functional.

There are no requirements about which specific settings we want to see as long as they are functional. Controls like volume, brightness, difficulty, graphics, or key bindings, would all be fine. A pause menu with the option to resume the game, change volume, and quit the game would suffice but we encourage more settings. The options or credits should include information about your team and your game, maybe a link to your TMS entry or to a blog, developer info, or whatever you like.

Through the menus, the game should save data in some form (RAM, local, or online) and allow the user to interact with this data. There is no requirement for the format of the save, or how many saves, just make sure that the game can access data from a previous play – whether a high score, settings, a full save state, or whatever your game saves.

For higher tiers, save something locally to a file, localStorage, or something similar. For the online requirement, integrating with Firebase, a database, or even a simple Node.js server that keeps track of data in an array are all valid methods. The server/database does not have to be persistent; we just need to see that data is being passed off to a foreign location where it will be held until requested in the same session. Don't worry if you are running your server on the same machine you play the game from. It's very common to have both client and server running on the same developer station when debugging.

Game Design

Design Doc

Write a design document. What are your milestones? What is being communicated to the player? What is the general direction and mood of the game?

Standard

- Team created a game design document that describes the game at a high level
- Design document lists milestones (that can hopefully be completed by the main event!)
- Design doc contains basic info such as win condition, genre, and target audience

Encouraged

- Design doc has been updated at least once to show changes in the development process
- Design doc has multiple milestones set
- Design doc communicates key game mechanics, the main character, and the setting

Extra Credit

- Design doc has been maintained through development with multiple updates and a changelog
- Design doc has UI mockups
- Design doc has all core game mechanics, all important characters, and info on all levels

Explanation

Design documents are an important tool in software development to help keep the team focused on a clear and unified vision. If one of your team members doesn't understand what the team is trying to make, it could lead to arguments or other slow-downs as you try to get everyone on the same page. Additionally, design documents focus the team and help drive everyone towards milestones which allows your team to demonstrate mechanics even before your game is completed.

Your design document should include a description of the game, some information on hardware requirements (PC, Ti84+, Xbox, etc.), development software (MonoDevelop, Visual Studio Code, etc.), and player point-of-view to name a few (first person, third person, etc.). Additionally, game style, target audience, and main plot points are important to decide and document before diving into development. There are formal standards for design documents (IEEE to be specific) but we won't be enforcing a specific standard. As long as you can show us that you had a document that had some design elements decided on before you started working, you'll get some points. If you're

lost, there is lots of help online about what a design document should look like!

As you work on your game, you'll probably find that things will change: requirements will be replaced or removed, your story will develop, and you'll probably miss some deadlines. For higher tiers of this achievement, show how your document changed, and covers most (if not all) of your game. The design document needs to be maintained, and complete!

Mechanics

Game tasks the player with completing various goals however the specific processes and environments in which the player accomplishes those goals can vary greatly.

Standard

- Player is given more than one mechanic for interacting with the game (for example, teleportation, rewinding time, picking up items, jumping, sprinting, etc.)
- Game has at least one, well-crafted and well-designed environment (for example, level, biome, map, etc.)

Encouraged

- Game has notably different mechanics (jump and double jump are *not* notably different)
- Game design describes multiple environments that showcase mechanics
  Examples:
  - Ice level that changes how the player moves
  - Dark dungeon that requires torches or light sources
  - Space station with low gravity

Extra Credit

- Choose at least one:
  - Game contains at least two sets of contrasting mechanics (enabling different play styles)
  - Game has at least one novel mechanic (other than move, jump, pick up, sprint, etc.)
- Game contains at least 3 environments that showcase the mechanics (levels, biomes, maps, etc.)
- Player can have a markedly different experience on each playthrough

Explanation

The main objective for this achievement is that your game has more than one setting for players to explore and more than one way to explore those settings. This variation helps

keep your players engaged through the whole experience of your game. Even the most interesting story can feel boring if only a single mechanic is used for hours on end.

Your varied environments could be independent levels, separate areas in one level, or the same area in a level but different textures and/or models. Mario Odyssey demonstrates this concept well by implementing desert levels (quicksand), ice levels (sliding with movement and needing cold weather clothes), and water levels (geysers). There are great examples of the environment impacting the mechanics. Various mechanics could be gliding, sliding, bribing, building/crafting, eating/drinking, keeping the beat or anything else you can think of.

For higher tiers, add more variation to the environment and mechanics but remember this is not a programming achievement. Showing the design and planning that went into a level that the programmers didn't have time to implement is ok within reason!

Target Audience

Who is the target audience/demographic? What concessions had to be made for them? Does the game adapt to the player?

Standard

- Game has at least one target audience (for example, kids like me, adults, my friends)
- Game has at least one feature or design consideration to better fit the target audience

Encouraged

- Game has at least one specific and well-defined target audience (for example, middle-aged grand strategy gamers, teenage casual gamers, elementary age Roblox fanatics, etc.)
- Team held design reviews to ensure game continued to be fun/engaging for the target audience
- Game has been playtested by the target audience(s)

Extra Credit

- Game has been playtested by members of the target audience(s) who are not team members
- Game mechanics, controls, color scheme, etc. all fit the needs of the target audience(s)
- Game has settings (difficulty, color blind mode, etc.) to better adapt to different audiences

Explanation

Every single game has a target audience: that is, the people the game was designed for. A few examples would be “elementary school students learning to type,” “commuters with long public transit rides,” or “11 to 18-year-olds who socialize through games." To make your game appeal to whatever audience you have chosen, you will probably have to make some decisions about what is appropriate for those players. For example, a commuter likely wants games that can be quickly picked up and put down without long play sessions or loading screens. In contrast, people trying to socialize in-game require a more immersive experience. Without keeping your target audience in mind, you might add features to your game that your players dislike. While there is no right or wrong in target audience choice, you need to make a convincing argument for your audience and related choices.

For higher tiers, make sure as much as your game as possible is built around your target audience. Simply saying you have a target audience doesn’t mean much if there are no design considerations that go into the game's implementation.

Evolution of Design

How was the game initially conceptualized? What changed when real people playtested? What was axed due to time or other constraints?

Standard

- Show meeting notes indicating changes in mechanics, assets, or other aspects of gameplay
- Game has been playtested (play from the beginning to the end)

Encouraged

- Have images showing early game builds with features, assets, characters, etc., that were dropped due to scope or time issues (take screenshots every now and then!)
- Team documents feedback while observing a playtesting session
- Team conducts a playtesting session and documents feedback and observations

Extra Credit

- At least one playtesting session was recorded (Screen capture is perfect)
- Game has been playtested by non-teammates on at least two separate occasions with team documenting feedback after each session

Explanation

Playtesting is an important part of any game creation process. It allows your team to verify that the game you designed is fun, engaging, and communicates what you were trying to communicate. Additionally, it helps identify where players may be confused or

lost, even if it was clear to you as the creator.

A good playtesting session should show you areas that need improvement, things you've done well, and any additions you may need and haven't thought of. It's a great way to get some feedback on your game before you have a panel of judges looking over your work and giving you a score!

For higher tiers, make sure to show how your game evolved. Playtesting sessions often provide useful feedback to shape your game and recording a session can highlight where the player had issues. Playtesting is also more useful if the person doing the playtesting is not on your team!

Communicate, Communicate, Communicate!

How does the game communicate with the player? How is the player taught new mechanics?

Standard

- Game has at least one indicator for the player's state (health bar, experience, luminance, resources, etc.)
- Game provides instruction on how to use new mechanics as they're introduced

Encouraged

- Game has at least two indicators for the player's state (health bar, screen fade, world assets, level progress, score, etc.)
- Game provides periodic feedback to the player (score, leveling up, etc.)

Extra Credit

- Game provides specific and continuous feedback to the player in more than two ways that aren't just numbers or bars (heavy breathing, camera effects, slowing walking speed when carrying too much, NPC reactions, etc.)
- Game indicates to the player when a mechanic is performed particularly well

Explanation

It is important for your game to effectively communicate information to your players so they understand how well they're doing. Health, money, and weight are all important pieces of information to the player but aren't usually visible in the game world. While simple numbers or displays may be sufficient to show these values to a user, more complex or subtle cues can provide a more immersive experience.

There are many ways to handle character indicators and player feedback. The easiest to

think about is a Heads Up Display or HUD (a health bar, map, selected ability, or sprint bar would all be examples of things you might find on a HUD) rendered on top of the game. Another option would be to adjust saturation levels as the player takes damage, gets tired/hungry/thirsty, or runs out of air, or remove saturation from the screen to indicate that there's a problem. Another method would be to have someone speaking to you over a comms device, have an NPC follow you around and talk, even put indicators on the player character themselves. We're just looking to see that the player is given some indication of what's going on in the game regardless of your specific implementation.

Similarly, your game needs to communicate effectively to players about what they can do and how well they're doing it. Your game should teach players about mechanics and how to use them as they're introduced. This usually means that you start the player off

with a few simple features/mechanics, allow the player to get used to them, and then add more until the player has full control of every mechanic (think Stardew Valley). In some games like Factorio, the features/mechanics are introduced via the skill/tech tree method. Any method may be employed, just make sure that players aren't overwhelmed when they first start playing your game and keep them informed on their progress!

For higher tiers, feedback provided to your player must be organic and not just another HUD bar, think crop sprites changing as they thrive or wither. In addition, the feedback should relate to how well the player is using the mechanics and provides indication or reward for excellent execution. For example, a perfectly timed double jump could create a special particle effect, maybe the player goes slightly farther, or even makes a "hya!" sound!

Art and Assets

Make Art

Make your game visually unique and give it your own flair!

Standard

- Create 3-5 custom visual assets
- Create a game logo and icon (not part of the 3-5 count)
- Cite all sources for assets not created by the team (those from creative commons, libraries, your game engine, AI tools)

Encouraged

- Create more than 5 custom visual assets
- Create title art or a splash screen (not part of the more than 5 count)

Extra Credit

- All visual assets made by team members
- Create store page background art

Explanation

Graphical assets include everything from character sprites to foliage models, particle effects to skybox/backgrounds, and even UI elements such as buttons. These assets do not necessarily need to be implemented into the game, but should be intended to be. Editing an asset made by someone else does not count, but, for example, making a texture for someone else's model will count toward this achievement (but not the other person's model). Remember, any assets not created by your team must be cited and you cannot pay for assets (nor commission) – there's no pay-to-win in OGPC!

In addition to the visual assets for the game, creating a logo, icon, title, and box art are great ways to get people excited for your game. Game logos and icons are emblematic of your game; they're relatively simple images that allow players to quickly identify your game. Think of the circle with an x used by Xbox or the flat s with a standing p used for PlayStation. Though often simple images, the creative process to create a logo and icon are often challenging and require multiple passes before everyone is happy!

Only assets created by the team will count for this achievement, however, any public domain or creative commons assets used must be cited and we will check your list!

For higher tiers, create more assets (rather than finding them in open-source libraries) or even all of your visual assets. Title screen or splash screen art are the first images a player sees when launching your game. These images are designed to get players hyped about playing your game and help it stand out from others. Store page background art is the art that would be highlighted on a digital store listing and often includes major

characters (and villains) and the title.

Hey DJ

Create a unique soundscape to provide depth and immersion for your game.

Standard

- Create one song (at least 30 seconds)
- Record at least one voice line or sound effect by a team member
- Cite all sources for assets not created by the team (creative commons libraries or your game engine)

Encouraged

- Create multiple songs (at least 30 seconds each)
- Record multiple voice lines or sound effects by the team

Extra Credit

- All music created by the team
- All sound effects created by the team
- All dialog created or recorded by the team

Explanation

The music and sounds of your game help draw a player in and create a layer of immersion that isn't possible via any other method. Hearing your character walk across leaves, hearing a light switch click, dings and bells for checkpoints or finding important objects, birdsong, or playing emotional music helps enhance the scene and pull your players in.

Only assets created by the team will count for this achievement, however, any public domain or creative commons assets used must be cited and we will check your list!

For higher tiers, more of the sounds and music in your game must be created by your team, and for the highest tier everything the player hears must be created by the team.

Cohesive Look and Feel

Everything fits together nicely, and nothing stands out.

Standard

- Have at least one color palette (consistent set of colors used everywhere)
- Have at least one soundscape (consistent set of sounds used everywhere)

Encouraged

- Assets are all of a consistent scale, resolution, and style; no assets are unintentionally jarring
- Sounds and music are a consistent volume, quality, and style; no sounds feel out of place or break player immersion
- Most player actions have sounds and animations associated with them

Extra Credit

- Audio and visual assets have depth that fit in with the game's setting and world
- Elements in the game's world have audible or visual cues to indicate interactivity or facilitate immersion
- All player actions have sounds and animations

Explanation

No one wants to play a game where half the assets are high resolution and interesting to look at while the other half are blurry and hard to identify. Similarly, nobody enjoys playing a game if the ambient sounds are quiet, but flipping a switch is so loud that you throw your headphones. While there is certainly a time and place for jarring visual or auditory experiences, it must be intentional!

In addition, color matters too. One (or more) color palettes can improve your player's ability to identify items in game and help scenes to feel cohesive. Choose a type of color palette (Monochrome, Analogous, Complementary for example) and be ready to explain your choice. Displaying your palette along with your concept art is a good idea and can help build a cohesive brand.

For higher tiers, make sure player actions have sounds associated with them and make sure all assets fit into your world and soundscape.

High Fidelity

Game uses particle/atmospheric effects and animated graphics to enhance the game.

Standard

- Create one particle or atmospheric effect (rain, fog, smoke, sparks, etc.)
- Choose one:
    - Create at least one animation sprite sheet (4+ frames)
    - Create at least one rigged and animated model

Encouraged

- Create 2+ particle or atmospheric effects
- Choose one:
    - Create 2+ animation sprite sheets (4+ frames)
      Create 2+ rigged and animated models

Extra Credit

- All particle or atmospheric effects created by the team to enhance the look and feel of the game
- All animation sprite sheets or rigged and animated models created by the team

Explanation

Have you ever sent something up in a satisfying puff of smoke, or had sparkles float around after you've cast a spell? These kinds of effects provide feedback for the player and make the game feel more immersive.

Animated graphics help make movements and actions feel more realistic. Animated graphics refers to objects and/or characters with multiple frames. Just moving a sprite is not sufficient. It could be a tree swaying back and forth in the background, a walking animation, a bird flapping across the sky, something growing or hopping, or twirling. They can be simple, as long as they have more than one frame.

It should be noted that any kind of gore effects or animations not only won't count toward this achievement but could disqualify your entry! Remember to keep your games E10+ qualifying.

For higher tiers, make more effects and animations though some can still be added from open-source libraries. For the highest tier, all effects and animations must be created by the team.

Biomes

Players are given multiple, unique experiences to explore.

Standard

- Create multiple levels/rooms/biomes/etc.

Encouraged

- Level/room/biome/etc. styles are consistent across asset types (visuals and audio)
- Assets all match the unique style of the level/room/biome/etc. (for example, spooky room plays spooky music)

Extra Credit

- Two or more modular assets each with multiple dynamic components that allow mixing on the fly to match player actions or locations

Explanation

The setting of a game is very important (especially when the theme could be interpreted as a setting) to build immersion and reinforce the story of your game. The player should be able to easily understand and feel immersed in whatever setting your game creates. Whether it takes place underground, in space, on a boat, or in a school gym, make sure the setting contributes to the player's experience.

Incorporating different environments into the story can be a bit challenging sometimes, but simply using varying backgrounds would not qualify. We're really looking for something where the story makes use of a new environment – think mountains vs plains, school vs at home, candy world vs wooden world. These different environments should potentially include different sets of sound effects or music to give each their own sense of style.

For higher tiers, not only should your game take place in multiple biomes, but some assets should be able to be mixed (by the programmers) to adapt to whatever environment the player is currently in or decisions the player has made. The emphasis here is assets that can be dynamically composed to make a wider variety or art (or music). For example, having a separate "muddy car sprite" and "clean car sprite" would not count. However, a base clean car sprite with overlay sprites for mud, snow, or damage would satisfy this, as would a character sprite to which different pieces of armor and tools can be overlayed.

Theme And Story

Use the Theme

OGPC provides a theme, you make it your own!

Standard

- The season's theme is present in the game

Encouraged

- The season's theme is clearly connected to the story and level design.

Extra Credit

- The season's theme is consistently tied to all aspects of the game – music, mechanics, setting, visuals, etc.

Explanation

The theme ought to be evident in every aspect of your game from sound effects to core mechanics. Game objects, dialog, sprites/models, music, and level design all need to be consistent with one another and relate to the theme. A cohesive theme ties your game together and makes it more interesting and immersive.

This is not to say that your game can't have aspects that vary from the provided theme, just be sure that everything is consistent within your game's universe. You can have lots of fun here, we encourage students to find creative interpretations of the theme, just be sure that everything is integrated and consistent. The game should be truly integrated with the theme across setting, story, music, visuals, and mechanics.

For higher tiers, tie the theme into most (or hopefully all) of your game. The theme should not be an interchangeable "coat of paint" on top of your game. Make sure it is well integrated into game design and mechanics.

Storyboarding

Plan out the theme, progression, player choices, and potential endings.

Standard

- Create a storyboard for at least one scene in the game

Encouraged

- Create a storyboard for the player's full path through the game (high level for the entire game)

Extra Credit

- Create *detailed* storyboards for all important moments in the game: opening, climax, key turning points, etc.

Explanation

A storyboard is a very important part of your design process and story development. A storyboard helps you plan how a scene looks, who moves where, and what parts of the story are progressed. Your storyboards should include descriptions that describe what is happening, and why the scene is important. The storyboards can be in your design document if you have one or in a separate document as long as they're all kept together.

For higher tiers, include more about your story. The more important moments you can include the better! For the highest tier, make sure that all of your important moments have detailed storyboards.

Story Telling

Every good story needs a cast and a good ending.

Standard

- Create a story outline that shows the general shape of the arc
- The story has multiple characters

Encouraged

- The game has a fleshed-out story with multiple sections
- At least one character is developed

Extra Credit

- Story has a clear beginning, middle, climax, and end.
- Story has at least one secondary or supporting character who is developed equally with the main character

Explanation

Everyone loves a game with a captivating story and a satisfying ending. Well-developed characters and fleshed out stories will draw your players in and make them want to come back to your game again and again. Planning ahead and tackling your story early in your game's development will help the team stay on track and help ensure all team

members are on the same page.

You should have an outline for your story which acts as an overview of your story including major plot points and other key decisions. There's no strict length or word count, as long as you've covered everything important and can walk a judge through your story.

Good stories also generally have multiple characters. Sometimes it's an old friend or enemy, maybe a love interest, or a close family member, but there should be at least one other character in the story. The who, what, when, where, and how of your story are completely up to you as long as your story starts somewhere, has some progression, and ends somewhere else, developing characters through your plot points.

For higher tiers, make sure your story builds towards a climactic event and finishes by concluding your major plots. Include a supporting character which your story spends some time on. They can't just be a quick blip on the radar, they need to recur and have a decent place in the story. To be clear, supporting characters are not an enemy type. They are characters in your story, they might be an enemy, they might be a companion, they might just be a narrative voice like in Bastion, but they need to have character development.

Scratching the Surface

Games are much more immersive when there is a deep story and fleshed out world to explore. Everything for this achievement must be able to be expressed via gameplay as dialog, narration, carvings, graffiti, audio logs, books, etc.

Standard

- Write a background for at least one location or character and explain how it is presented to the player
- Write at least one world event that happens prior to the game starting and explain how it is presented to the player

Encouraged

- Write a background for multiple locations or characters and explain how they are presented to the player
- Write multiple world events that happen prior to the game starting and explain how they are presented to the player

Extra Credit

- Tie characters, world events, and location histories into at least one playable subplot (even if it doesn't make it into the game)

Explanation

Players love games with rich lore and an expansive universe to explore. Hints of the universe continuing around the events of your story help to build immersion and keep players coming back to discover more.

Think about the characters in your game and write some detailed backstories for them. Consider where and when your game takes place and write some history. Think about how you want to portray these histories to the player, whether books, notes, or graffiti, and make sure the player is capable of piecing together at least a little bit of your stories!

For higher tiers, create an entire subplot based on some of your world history. This could be a story based around someone who remembers the world before it turned into cheese, or a fetch quest given by someone looking for the ancient socks worn by a hero of yore.

Fan Fic

Flesh out the world and story in materials not designed to be integrated into the game itself.

Standard

- Create one story in your world that is not present in the game. This story should tie into a character or location that is in your game and can be any length
- Create a glossary of characters and locations

Encouraged

- Create an *illustrated* glossary of all characters and locations including descriptions and relevant backstory

Extra Credit

- Create a reference guide (printed or uploaded to TMS) for important elements of the story including:
    - Characters and locations that may not be implemented in your game
    - Includes the illustrated glossary of all characters/locations
    - Ability, power, and item descriptions

Explanation

When players get attached to a game world and characters, additional stories and reference materials can help players maintain a high level of engagement even outside of gameplay. Write a story that expands on events, locations, or characters from the game to provide additional backstory. This should be designed to be read more like a book and

not to be consumed by a player while playing your game.

In addition, consider all of the major characters and locations in your game and write/compile a glossary of everything and everyone related to your game.

For higher tiers, create a reference guide to the story, background, and characters present in the game. This reference guide should not be your design doc, this should be a fun supplement to your game that a player would be interested in exploring. The goal is to provide additional context and flavor for your game, as well as expand on details that didn't quite make it into the playable game itself.

MANAGEMENT

Team Players

The team got along nicely, communicated well, and overcame challenges together.

Standard

- Team members met (in person or virtually via Zoom, Hangouts, Discord, etc.) at least once a month for a group work session
- Team describes a challenge they faced while working together as a team

Encouraged

- Team members met (in person or virtually via Zoom, Hangouts, Discord, etc.) regularly for a group work session (at least monthly) and record what was worked on

Extra Credit

- Team scheduled review sessions and weekly check-ins with documented notes/minutes
- In the "Making Of" video, the team discusses how work was divided and each what role each person took on the team

Explanation

Communication as a team is vital to creating a positive team environment as well as a polished game. Having trouble figuring out what to work on next? Send a message or email out to the group. Not sure how this bug is happening, or why Gimp isn't working? Send out a message to everyone. Found a cute cat picture you want to share with everyone? Make sure your whole team gets to see it! There are many free tools you can use such as Slack, Remind, or even just email. Pick a tool and use it through the development cycle to stay connected with your team.

For higher tiers, make sure you communicate as often as possible. Using review sessions and weekly check-ins can help keep the team organized and on schedule and answer questions that might be blocking progress.

Public Relations

Be professional! Be ready to pitch your game and make it memorable.

Standard

- Team members all wear coordinated clothing (printed, matching colors, or matching styles / themes)
- Appoint a team spokesperson who is ready to introduce your team and present your “elevator pitch”
- Create a banner or flag that is on your table that identifies your team

Encouraged

- Create a development blog/vlog/social media page that contains 5 or more posts focused on at least two different aspects of the development process (coding, art, etc.)
- Create some swag for giveaways (buttons, pins, magnets, brochures, merch, etc.)

Extra Credit

- Development blog has multiple entries by all members on the team spanning several months

Explanation

You know what looks cool? A whole team in matching outfits! While some teams do make T-shirts for their team, you do not have to buy new clothes for your team to look coordinated. You can all wear jeans and black t-shirts, or dress pants/skirts with white shirts. You could even all dress up like characters in your game. You decide on your team’s look and make sure everyone matches.

Your TMS page is great as your official download/details page, but you can really reach out to your fans with a blog, Facebook page, or other social media. Use your logos and pictures, and publish information about your team process. You could even use it to keep your meeting notes (minutes). Fans always love to see the process of how their favorite games are made.

You don’t have to spend much money to really bring the team spirit. You can paint or cut out a banner, you can design a brochure on the computer, or even have fun with handmade pins! Big shows like PAX and E3 are full of handouts to get people to notice the games. Screenshots, descriptions, team information, game logo and your team logo are all good things to add to your items.

For higher tiers, maintain your blog, and create some physical items to build hype for your game. Don’t just create a blog or social media page and abandon it! Go back to it to post a new status or other details every few days for a couple of months. Don’t let down your fans!

Team/Project Management

Use tools to help manage work: issue tracking, milestones, task management and more.

Standard

- Team uses some form of task management tool (Trello, Jira, Asana, Excel, GitHub Projects)
- Team is prepared to present a dashboard (or equivalent) during judging

Encouraged

- Task management uses some form of grouping (tasks and subtasks)
- Tasks are assigned owners and state is tracked as work is completed
- Include bug management with tasks

Extra Credit

- Task management is broken into sprints or other milestones
- Add weights/priority to tasks

Explanation

Project management isn't easy, but it's a crucial step in keeping a game on schedule and having it ready to compete with at the Main Event. Tools such as Trello or Jira (feel free to use whatever tool works best for your team) can make this process easier/faster and having everything in an electronic format makes it easier for everyone to be checking in even if they're working on tasks from home.

For higher tiers, group your tasks (the more tiers the better), track completion of individual tasks and sprints, use weights on your tasks, and track your bugs. This might seem like a lot at first, but it is how real-world projects are managed! The grouping of tasks, weighting of tasks, and tracking of tasks makes sure that work gets done and that nothing is being forgotten about. This process also helps prioritize which tasks to take on next to make sure your team is on track. Generally, there are fewer milestones than there are tasks: think of "complete all character animations" or "complete rough draft of story" as milestones and "make a walking animation" or "record a dialog line" as smaller tasks. Assign time estimates for all of your smaller tasks, add them up to see how long you expect your milestones to take.

Checking the Boxes

Complete TMS. All of it. I dare you.

Standard

- Fill out at least these fields on TMS:
  - "About this Entry"
  - Game name
  - Team member list
- Include a basic description of the game
- Leave no remaining placeholder images (team members, team logo, game poster)

Encouraged

- Fill out all game entry page details on TMS
- Include a playable link or download link to your game
- All team members are linked to game entry
- Include more than one sentence for the About section and the Instructions section

Extra Credit

- Game entry on TMS has 10+ images (screenshots, concept art, development images) and all images have descriptions
- Include a full game description (200+ words, think Steam store page), instructions, etc.
- All team members have appropriate roles set in TMS and have profile images, and a team group image is included

Explanation

The Team Management System at https://tms.ogpc.info/ allows you to publish your game to the world on the OGPC site. It provides a form for you to describe your game, your team, and how your game was created. Best of all, it allows others to play your game and check out other games.

No game on Steam or Epic would sell well without all the details filled out. TMS has slots for team logo, game logo, team group photo, and game screenshots. Make sure you add logos/screenshots/images wherever there's space! Not only is it worth the achievement, our awards presentation gets generated directly from TMS, so you don't want to be the team with "NO PICTURE" next to its name!

For higher tiers, you've got space for pictures on TMS, but you can also upload your game or link to it so people can play it. It's so much fun to be able to play everyone else's game, so make sure you give teams the chance to play yours. Fill out the game description and instructions, set the game engine and language, and make sure that you

fill out everything. Use the “Edit Game Entry” in the upper-right of your game details page. If there’s a blank spot on TMS, fill it! Upload as many images as you can and add descriptions. This all helps judges and other teams to get as much info about your game as possible before playing it!

Production Value

Trailer and "Making Of" videos are well put together, and describe the game, team, and development process.

Standard

- Trailer and "Making Of" videos are submitted to TMS
- Videos are scripted, well-practiced, and within the time limits (see Competition Manual)

Encouraged

- Trailer and "Making Of" videos have well-balanced, normalized audio (no one is ear-blastingly loud or unintelligibly quiet)
- Videos include in-game footage with a voice-over
- Videos are edited (no mistakes, blunders, or awkward gaps)

Extra Credit

- Create at least one additional trailer, promotional video, or behind the scenes video (at least 1 minute long)
- All required videos have *reasonable* special effects (titles, fades, transitions, etc.)
- Videos are designed with a storyboard

Explanation

Do you drool over videos for the newest games on Steam and YouTube? It’s time to make your own! Capture some gameplay, add some titles, maybe even create a custom scene. Watch a few game trailers to see what you like about them, and then make something similar for your game. Some movie maker software even makes it easy to combine clips with music in different styles. Make sure to use music and voice clips that you have permission to use (same rules as your game assets) and don’t forget to cite them at the end. Even better, use your own content!

For higher tiers, make sure that everything fits together well, use clips of gameplay from your videos with your voices, and even use a storyboard for the videos just like for your game’s story. If you feel really ambitious, make an additional video for your game in a different style than the required videos. Trailers, promotional videos, and behind the scenes videos generate hype and help draw people into your game. Everyone loves learning a little more about their favorite games!

Changes and Amendments

The Oregon Game Project Challenge reserves the right to change, amend, modify, suspend, continue, or terminate any or all rules and regulations of the main event, including achievements, either in an individual case or in general, at any time without notice.

Change Log:

November 2022 – First release updated for 2022-2023 challenge year