Milestone Submission Form

Team ID: 9
Milestone: 4

Instructions

Please keep this document up to date and include a copy of it with each milestone submission. TA's should be able to read through this document and see the work your team has completed for the current milestone as well as a running history of work completed over prior milestones.

Milestones are organized into tables below which include both their required features and text inputs for their creative components (which you fill in with your selected features). There are annotated long-form explanations for each feature, with links to additional resources if available, at the end of this document. We further provide a table of suggested features that you may pick and choose from for each milestones creative component. To aid in planning, each suggested feature has a classification of 'basic' or 'advanced' and a list of the background knowledge required to implement it. In general, 'basic' features are worth 10 points and 'advanced' features are worth 20 points. We highly encourage you to work on your own custom features, beyond what has been suggested here. Important: Please discuss the amount of points custom feature are worth with the TAs before working on them.

Fill in the table below for the current milestone, entering the initials of the author responsible for each implemented feature to the right of the table, under 'Author'.

Grading

Each feature you implement allots your team a specified number of points, for each milestone you must attain 100 points to receive full marks for that milestone. Certain features are required for specific milestones, others you can complete as as a part of the creative component of milestones. Required features can be completed early but never late, meaning if you finish the required features for future milestones in an earlier milestone submission, you will be credited at the earlier milestone, leaving further milestone, leaving room for additional optional features in the future one.

You will receive full credit for features only if they are fully operational. We deduct points for sloppy, buggy and incomplete implementations. Grading suggested features will necessarily be subjective: more complex features or those better fitting into the overall game will be rewarded with more points. Bonus points can be gained for features exceeding 100 points, and the grading of additional bonuses, features, and the size of bonuses will be at the marker's discretion.

Milestone 1

Category	Task	Points	Author
Rendering	Textured geometry	10%	KC
	Basic 2D transformations	10%	KC
Gameplay	Key-frame/state interpolation	10%	Team effort
	Keyboard/mouse control	10%	Team effort
	Random/coded action	5%	ST
	Well-defined game-space boundaries	5%	KC
Stability	Correct collision processing	10%	SB, CF
	Wininal lag	20%	Team effort
	No crashes, glitches, unpredictable behaviour		Team effort
Creative	Gamepad & controller input manager	20%	KC
	2D camera that follows player around		ST
	Enemy and health components (take damage when walk into an enemy)		SM

Milestone 2

Category	Task	Points	Author
Improved Gameplay	Game logic response to user input	20%	CK, KC, CF
	Sprite sheet animation	15%	ST
	New integrated assets	10%	SB, KC, ST
	Basic user tutorial/help	5%	KC
Playability	2 minutes of non-repetitive gameplay	15%	Team effort
Stability	Wininal lag	15%	KC
	Consistent game resolution		KC
	No crashes, glitches, unpredictable behaviour		Team effort
Creative	Added self-made mutation assets	20%	SB
	Procedural platformer level generation		KC
	Sprite batching		KC
	Parallax scrolling background		ST
	Enemies projectile attack (fireballs)		SM

Milestone 3

Category	Task	Points	Author
Playability	5 minutes of non-repetitive gameplay	15%	Team effort
Robustness	Memory management	10%	Team effort
	Handle all user input	5%	Team effort
	Real-time gameplay	10%	Team effort
Stability	Prior missed milestone features & bug fixes	20%	Team effort
	Consistent game resolution		Team effort
	No crashes, glitches, unpredictable behaviour		Team effort
Creative	Paralax background in main menu	40%	SB
	Enemy spritesheet assets and animations, death animations		SB, ST
	Extendsible Player Mutation System + Mutation Selection UI		KC
	Bow animation and projectile physics		CF
	Coins and exp drop after enemy death + part of the animation		SM
	multiple different pathing modules based on A*, patrolling behaviors		CK
	easy mode (standard / hard difficulties) to account for differences in players		CK
	In-game shop to buy additional upgrades		ST

Milestone 4

Category	Task	Points	Author
Stability	Prior missed milestone features & bug fixes	15%	Team effort
	No crashes, glitches, unpredictable behaviour		Team effort
Playability	10 minutes of non-repetitive gameplay	15%	Team effort
User Experience	Comprehensive tutorial	10%	
	Optimize user interaction and REPORT it	10%	
Creative		50%	

Suggested Features

Category	Feature	Group	Background Knowledge
Graphics	[1] Simple rendering effects	basic	Fragment shaders/OpenGL uniforms.
	[2] Parallax scrolling backgrounds	advanced	Vertex and fragment shaders, texture mapping.
	[3] Complex geometry	advanced	Geometry and vertex shaders.
	[4] Skinned motion	advanced	Meshes, bones, constraints, matrix algebra and hierarchies, UV mapping, kinematics.
	[5] Particle systems	advanced	Instanced rendering, shader storage buffer objects, simple physics.
	[6] 2.5(3D) lighting	advanced	Normal mapping, local illumination models
	[7] 2D dynamic shadows	advanced	Basic shadow mapping, ray-object intersections
Physics & Simulation	[8] Basic physics	basic	Basic understanding of 2D physics.
	[9] Complex prescribed motion	advanced	Bezier/spline/Hermite interpolation and parametric curves.
	[10] Precise collisions	advanced	Newton's method, basic physics, acceleration structures such as bounding volume hierarchies, quad trees, etc.
	[11] Complex physical interactions with the environment	advanced	Classic physics models, kinematics, numerical integration.
	[12] Articulated motion	advanced	Parameterization, kinematics, coordinate systems, matrix algebra and hierarchies.
	[13]Physics-based animation	advanced	Classic physics models, Euler method or other advanced integration methods, kinematics, particle systems for background effects (water/smoke).
AI	[14] Simple path finding	basic	Basic search algorithms (ex. breadth-first).
	[15] Advanced decision-making	advanced	Complex graph traversal and search algorithms, goal-based AI logic (ex. rewards, penalties...).
	[16] Swarm behaviour	advanced	Instanced rendering, BOIDS, basic physics.
	[17] Enemy group behaviour Cooperative planning	advanced	Behaviour/decision trees, observer pattern, BOIDS.
	[18] Cooperative planning	advanced	Behaviour trees, goal-based AI logic (ex. rewards, penalties...), observer pattern.
Software Eng.	[19] Reloadability	basic	Serialization.
	[20] External integration	basic	General coding skills.
UI & IO	[21] Camera controls	basic	Linear algebra for camera matrix.
	[22] Mouse gestures	basic	General coding skills.
	[23] Audio feedback	basic	General coding skills.
Quality & UX	[24] Basic integrated assets	basic	Asset creation tools (e.g. Blender, Krita, GIMP, Audacity...).
	[25] Game balance	basic	Video games, human psychology :)
	[26] Numerous sophisticated integrated assets	advanced	Asset creation tools (e.g. Blender, Krita, GIMP, Audacity...).
	[27] Story elements	basic or advanced	Narratives, basic animation (for cutscenes), text rendering, or text sprites.

[1] Simple creative use of the fragment shader. For example changing the color of a sprite over time. The color should change based on a uniform input (e.g. change the uniform based on time, user input or when a collision is detected).

[2] Multiple background layers (at least 3) that create a parallax effect upon camera motion.

[3] Incorporate one or more complex polygonal geometric assets. Implement an accurate and efficient collision detection method that supports this and other moving assets (include multiple moving assets that necessitate collision checks).

[4] Render an animated skinned mesh (for example an eel represented as a triangle mesh that slithers around).

[5] Use the OpenGL instancing feature glDrawArraysInstanced to render hundreds of instances of the same object more efficiently to create appealing particle effects.

[6] Create interesting shading effects, such as diffuse reflection, metallic texturing, bump/normal mapping, specular reflections, baked/static shadows...

[7] Make lights cast dynamic shadows, when entities pass in front of a light source their shadow should be accordingly updated and look plausible/realistic. You can use any technique as long as the result looks good.
https://www.gamedev.net/tutorials/_/technical/graphics-programming-and-theory/dynamic-2d-soft-shadows-r2032/

[8] Simple physical interactions, force of gravity, elastic or inelastic collisions, conservation of momentum... For example, have a ball fall down and bounce off the floor/entities. Use a numeric integrator such as Verlet.

[9] Use geometric splines (Hermite, Lagrange, Bezier, etc.) to implement smooth non-linear motion of one or more assets or characters. An example of a curve controlled animation is shown at
https://docs.unity3d.com/uploads/Main/AnimationEditorBouncingCube.gif

[10] Incorporate two or more complex polygonal geometric assets that move and collide. Implement an accurate and efficient collision detection method that supports these and other moving assets. You can approximate all objects with convex proxy polygons.

[11] Have complex physical interactions between the entities and the environment. For example simulate exact collisions between the player and ropes/vines that wiggle and eventually come to rest, or let the player cut a tree and have it fall down in a realistic/plausible fashion.

[12] Implement an articulated entity, for example a robotic arm that follows the cursor and uses inverse kinematics to figure out its position/geometry. Use a matrix hierarchy and correctly solve the inverse system.

[13] Implement time stepping based physical simulation which can either serve as a background effects (e.g. water, smoke implemented using particles) or as active game elements (throwing a ball, swinging a rope, etc.). A subset of the game entities (main or background) should possess non-trivial physics properties such as momentum (linear or angular) and acceleration, and act based on those.

[14] Breadth first search for path finding and logic for characters to follow a prescribed path.

[15] Advanced decision-making mechanisms based on goals (e.g., A* result used in the AI of a character).

[16] Create a group of characters with entities of the same class influencing each others positions. Examples:
Subsaatva's fish schools: https://eater.net/boids
BOIDS pseudocode: http://www.kfish.org/boids/pseudocode.html

[17] Have a group of enemies coordinate between them, for example have them create an organized line to pass through a bottleneck to reach the player, have a healer enemy heal an ally when they get wounded, have many enemies position themselves to best surround and block the player out of an objective.

[18] Planning the action of two different characters towards a common goal that requires non-trivial communication between the two (e.g. coordination between non-player characters and enemies towards a joint goal).

[19] Write level descriptions (entities and their components, i.e., position, texture, ...) in a human-readable text file and write a level loader. We recommend the JSON format using existing json loaders. ECS makes it easy to add components programmatically. The game should allow for full state saving for play reload. Users should be able to exit the game and restart at the same place they left the game, with all environment variables reset to the state they were in at save time (unless some variable needs to be reset to make sense in your game).

[20] Integrate one or more external tools or libraries (physical simulation (PhysX, Bullet, ODE, etc.), EnTT ECS system, game engines, or other alternatives). Important: Make sure that the installation works for all team members before merging to main. It was a major issue in the past that teammates were not be able to contribute and test the program due to a different operating system or development environment.

[21] Make the camera follow the player or move it based on mouse or keyboard input.

[22] Recognize gestures (patterns drawn with the mouse) to trigger jumps along an arc or other dedicated action.

[23] Add audio feedback for at least three interactions in the game as well as background music with tones reflecting the journey of the game.

[24] Create a few additional assets, such as new sprites and fully integrate them into the game, either as background elements or as interactive entities.

[25] **Do it only for the last milestone.** Make sure your game is balanced and fun to play, your game should be beatable but should still require some level of challenge to the player.

[26] Have complex assets, such as music that changes when the player is in/out of combat, animated meshes (e.g. gltf files), a wide variety of visually coherent sprites (i.e. same aesthetic/artistic style)...

[27] Give a compelling story to the game. Have some basic character development and interesting events. You can either lean more on artistic creativity (e.g. interesting story/plot) or technical merit (e.g. complex cutscenes).