Grading guidelines

1. Content

Grading guidelines 1
1. Content 1
2. General 1
3. Game grading 2
4. Code grading 3
5. Code insight grading 3

2. General

This document gives information about the grading criteria that will be used for both the milestone and deadline. It is important that you carefully read this document before making a game choice, because the chosen game should contain the elements that are graded. For example, if the game has no camera, you get no points for the camera grading part.

3. Game grading

You cannot get a column-grade when the conditions in the column(s) to the left aren't met. E.g.: You can only get to the **Meets Standards** column when you also meet the requirements of the columns to the left of this, like the **Basic** column.

	Insufficient	**Basic**	**Meets standard**	**Exceeds standard**
Camera **+**	- The camera automatically follows the main character or another item or based on input. - The camera sometimes shows parts outside of the level boundaries. - Camera jumps to different rooms.	- The camera never moves outside the level boundaries. - Camera makes a transition move to different rooms/areas.	- Free moving area: when the avatar moves a limited distance, the camera doesn't follow the avatar. - Parallax background effect. - Ease in/out camera moving speed. - Camera automatic zoom in/out	- Animation system based on data from a file. - load/save game system - load level from a file - Bone animation - Pathfinding - A.I. - Inventory system. - Level editor ... - Particle system - Level generation - others…
Animations **++**	Only a few simple animations comparable to the minigame animations.	There are a lot of animations on more than one game object. All the animation loops are smooth without hiccups or frame jumps.	There are animations that are triggered by game events or other game objects. e.g.: - Dying animations. - Contact animations - Grabbing a ladder, a ledge, …	
Interaction player and props, enemies, level, equipment **+++**	Only interactions with no influence on the state of other game objects, e.g..: coin pick up	Interactions between different game objects that influence each other state.	- Interactions between two (or more) different game objects using a third one. that influence each other state. e.g. destroying game objects using projectiles or tools. - Transfer ownership of one object to another	
Game implementation (how much is there) **++++**	Very simple gameplay. e.g. running game where there are only coins to collect such as the minigame.	A limited number of game object types with simple behavior.	Complicated gameplay with many different game object types.	
HUD and UI (heads up display and user interface) **++**	A **simple** UI or HUD which is implemented using textures, e.g. - texture based font - textured UI or HUD	A **complex** UI or HUD with multiple elements and includes sprite sheet animations.	Menu system, not just a start screen, but a menu that can be called and changes game settings such as sound on/off and other settings.	
Sound **+**	Ambient or environment sound is present.	Some actions in the game have a sound representation.	Many actions in the game have a sound representation that is played / canceled when necessary. The volume can be adjusted/turned on/off using keypresses.	

4. Code grading

When the project contains only **simple basic code** then you'll get **Insufficient** for this grading part.

Design, structure and overall quality	1. Classes and functions have a clear and single responsibility and work as independently as possible (decoupling). 2. Good use of composition, inheritance and polymorphism. 3. Encapsulation, correct use of access specifiers. 4. Correct use of static members if there are any. 5. Using class enumerations and structs when applicable. 6. Use of advanced techniques (techniques not seen in class) when appropriate.
Penalties	1. Not applying the coding standards as specified in the standards document: • Not applying the naming conventions correctly • Choosing not self-explanatory variable and function names • … 2. Bad low-level structure (within functions and code blocks) 3. Making code more complex than it needs to be. 4. Using hard coded values. 5. Bad implementation of the rule of 3/5. Copy and move semantics. 6. No const correctness of member functions. 7. Not passing parameters by (const)reference/pointer/value where appropriate. 8. Poor performance. 9. Exceptions or crashes. 10. Memory or resource leaks. 11. Dynamic memory allocations in the game loop, every frame 12. Base class destructor not virtual when the class is not final. 13. Bad polymorphism implementation (e.g. derived members/functions that should be moved to the base class) 14. Excessive use of singletons thus hiding bad class design 15. Code that is not in the fitting class. E.g. many getters and setters used to modify members from outside the class …

5. Code insight grading

When the project contains only simple basic code then you'll get **Insufficient** for this grading part.

Insufficient	***Basic***	***Meets standard***	***Exceeds standard***
Only has a conceptual understanding. Not able to answer some questions about the code. Not able to explain the code.	Understands the code superficially and can explain/answer what was asked without many details.	Understands the code thoroughly and can give a very clear explanation with lots of details and can offer other strategies to solve potential problems.	Can offer other not so obvious strategies to solve potential problems.