EXORDIUM GAMES

Programmer assignment: Inventory system

Instructions

This task comes with a lot of optional ("Bonus") features. They are marked with the starting “Bonus” text after the feature number. If a feature does not have the “Bonus” text after the number, even between “Bonus” features, it is an obligatory feature. Each additional Bonus feature will be appreciated and valued.

Syntax instructions:

- Fields and variables
  - Public – use CamelCase notation
  - Protected and private – use _variableName notation
  - Local variables – use variableName notation
- Methods
  - always CamelCase notation
- Enums – use enum EnumNameType{ EnumValueA, EnumValueB} notation
- Constants – use CONSTANT_NOTATION

Unity instructions:

- Use Unity version 2019.2.17f1

Notes

- there is no need to waste time drawing your own inventory items/character
- you can use any graphics you find useful from online or any other sources
- The executable file will be tested on 16:10 and 16:9 aspect ratios, and must work (UI must scale) on all horizontally oriented aspect ratios of the window
- Advice: check the accessibility, functionality, and testability of all the data you send from another computer (builds, means to access repository)

IMPORTANT!
In your response mail you should provide the link to a google drive **folder named “Exordium_ProgrammerAssignment_dd.mm.yyyy”** with the following content:

- **“Exordium_ProgrammerAssignment_Level1_InventorySystem_dd.mm.yyyy.docx”** - a copy of this document, named”, with a green highlight of the text for each point you solved completely, yellow for partially, and red for unsolved.
- **InventorySystem.zip** containing a .zip of the "Assets" and "ProjectSettings" Unity project folders
- **Folder “Builds”** containing the following:

  - **InventorySystem_64.zip** containing the windows 64-bit standalone exe build and data folders
- **InventorySystem_README.txt** which explains the testing and grading guidelines for your solution - buttons and input mappings, and which script is relevant for each subtask point.
- In your solution, in the InventorySystem_README.txt, provide the link and credentials required to access the online repository / or add the following mail to have the access to the repository exordium.sandbox1@gmail.com and mention it in the file - more details are provided in the Subversioning Development task.

For all questions and unclarities about the assignment, contact andrija.stepic@gmail.com

1. 2D player moves on the XY axis in Unity world space (example: based on WASD and/or arrow keys input) If the player presses both horizontal and vertical input, the movement direction is normalized.

1. Movement in 2D XY axis example image - Pokemon

✓ 1.1. 2D physics-based movement instead of transform.position based
✓ 1.2. At least 4-directional sprite sheet animation
✓ 1.3. Animation speed matches the movement speed
✓ 1.4. The camera must follow the player in 2D space
✓ 1.5. The world must contain at least one object collidable with the player

2. Player character can pick up items from the ground to the inventory

Choose any of the following options:

✓ 2.1.1. Option 1: Pick up items by clicking on the item in predefined spatial proximity

✓ 2.1.2. Option 2: Pick up items by physics trigger collision - continuous

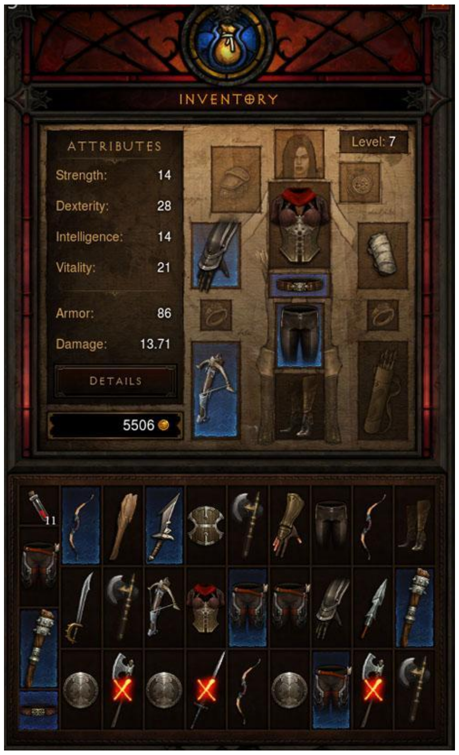
✓ 2.1.3. Option 3: Pick up items by player position + physics overlap circle - continuous

✗ 2.1.4. Option 4: Pick up items by physics circle casting in movement direction - continuous

✓ 2.1.5. Bonus: any option beyond the first and the possibility to switch between them during runtime.

If you choose the option of using physics, items should be triggers, not colliders.

Inventory consists of these 2 parts:
1) Inventory Grid screen
2) Equip screen



2. Inventory grid screen (bottom) and Equip screen (top) example image – Diablo 3

Note: although the screens on Diablo 3 example image, shown above, are combined in one, this assignment requires separate windows for inventory and equipment.

EXORDIUM GAMES

3. Opening/Closing inventory and equipment screen

Note: All UI elements in the entire assignment must be achieved by using Unity 4.6+ UI, not Unity's OnGUI API, nor external plugins such as NGUI.

✓ 3.1. Button for opening the inventory screen. It is shown when the inventory is hidden, and hidden when the inventory is opened.

✓ 3.2. Button for closing the inventory screen as a part of the inventory screen.

✓ 3.3. Bonus - inventory screen can be opened/closed with a shortcut key (like "I")

✓ 3.4. Button for opening the equip screen. It is shown when the equip screen is hidden and hidden when the equip screen is shown.

✓ 3.5. Button for closing the equip screen as a part of the equip screen.

✓ 3.6. Bonus - equip screen can be opened/closed with a shortcut key (like "E")

4. Inventory Grid Screen

You should implement only 1 level of the inventory grid screen. The higher, the better.

✓ Level 1: fixed grid cell count visible at all times (minimum of 4x8 tiles) - see Diablo 2 game inventory screenshots

✓ Level 2: fixed grid cell count, with scroll rect and X rows visible at all times (minimum of 4x8 tiles, example: grid consisting of 20x8, with 4 rows visible)

– Level 3: dynamic grid with flexible cell count - rows are added by the need and never removed (example: grid starts with scroll rect of 4 rows with 8 cells each, on picking up the 33rd item, the grid expands by one row)

✗ Level 4: dynamic grid with flexible cell count - rows are added and removed by the need (example: grid starts with scroll rect of 4 rows with 8 cells each, on picking up the 33rd item, the grid expands by one row, upon removing the items from that row, the row disappears)

5. Equip Screen

✓ 5.1. Must have only fixed cell slots

✓ 5.2. Have at least 4 different cell slots for items (example: head, torso, weapon/main hand, shield/off-hand)

6. Inventory functionality

✓ 6.1. Picked up items can be previewed inside an inventory UI grid

✓ 6.2. Bonus - if the Equip Screen slot for the picked up item type is empty, that item is automatically equipped in that slot of the Equip screen, not placed to the grid (example: you pick up torso armor for the first time, and it gets equipped to the torso slot)

✓ 6.3. If the Inventory Grid type is not level 3, nor level 4 (infinite), if there are no more free grid cells, the item is not picked up

✓ *6.3.1. A suitable debug message is reported*

✓ *6.3.2. A suitable message is presented to the user*

7. Item Interaction

✓ 7.1. Left click on the item in the grid or the equip screen forces the clicked item "into the air" and the item follows the mouse screen position

✓ 7.1.1. If the item is “in the air” and left mouse click is executed over inventory or equipment window, if the slot is of the appropriate type (equipment slot type), or it is an inventory slot, the item is slotted in the new cell

✓ 7.2. If all opened inventory screens get closed, and an item is "in the air", the item is returned to its previous position/slot

✓ 7.3. If a left click/touch is pressed outside the screens while at least one screen is opened and the item is "in the air", the item is dropped to the ground next to the player

✓ 7.4. Right-click on an equipable item in the grid equips that item in the slot of the equip screen. If there was an item in that slot, it is moved to the slot in the inventory grid screen.

✓ 7.5. Right-click on an item in an equip slot unequips that item to the slot of the inventory grid screen, if there is any. If none, nothing.

✓ 7.6. Mouse + key combo input shortcut on an item drops the item to the ground next to the player

✗ 7.7. Bonus: mouse hovering + key shortcuts for dropping, equipping and unequipping an item

✓ 7.8. Bonus: item highlight upon hovering before pickup in world view

✓ 7.9. Tooltip window for item’s properties (example stats: name, type, stats, image) in inventory and equipment windows upon hovering over the cell containing an item

✓ 7.10. Middle mouse button click on the consumable item consumes the item.

✗ 7.11. Bonus: Additional button as a part of inventory panel, which sorts items by type, and condenses stacks of the same item type.

8. Items

✓ 8.1. Each item should have a string variable - name of the item

✓ 8.2 There are 2 main types of items

✓ *8.2.1. Permanent usage items - applied upon pick up, not picked up into the inventory (the suitable Debug.Log message with the item's name is sufficient)*

✓ *8.2.2. Pickup-able items - can be picked up into the inventory*

✓ *8.2.2.1. Equipable items*

✓ *8.2.2.1.1. Equipable items have a variable representing the slot type of the equip screen this item can be equipped to.*

✓ *8.2.2.2. Non-equippable items*

✓ *8.2.2.2.1. Stackable*

✓ 8.2.2.2.1.1. Stackable with stack limit (eg. 4)

✗ 8.2.2.2.1.1.2. Bonus: Change color based on the percentage of items until the stack is full (At least 2 colors – full/not full)

✓ 8.2.2.2.1.2. Stackable without stack limit (max int)

✓ 8.2.2.2.1.3. While the stackable item is in the air, remove the stack display from the inventory slot, it must move with the dragging item

✓ *8.2.2.2.2. Non-stackable*

✓ 8.3. There must be an option of spawning new items to the ground, indefinitely. Bind it to a key (for example Space) or an UI button.

✓ 8.3. There must be at least 1 stackable limited, 1 stackable unlimited and 1 useable item, along with 4 equippable items

✓ 8.4. IMPORTANT: Item data must be inside a serializable struct/class or a scriptable object, and inventory / equip slots must have instances of items that hold the item data.

✓ 8.5. Items must provide permanent attribute bonuses to the character while they are equipped which must be visible on the stats screen

9. Attributes and Attribute UI window

✓ 9.1. The player must have at least 4 types of attributes. Example: Strength, Dexterity, Agility, Intelligence. The attribute is represented by the pair of the attribute type and an integer value.

✓ 9.2. Player’s attributes must be clearly visible on Attribute UI window as a list of pairs: (attribute name, integer value)

✓ 9.3. Button for opening the attribute screen. It is shown when the attribute screen is hidden and hidden when the attribute screen is opened.

✓ 9.4. Button for closing the attribute screen as a part of the attribute screen.

✓ 9.5. Bonus - attribute screen can be opened/closed with a shortcut key (like "C")

10. Subversioning Development

- Create a private online repository
- Develop the project in Unity so all the minimum relevant files are committed and pushed to the repository
- Use branches for separate features
- Maintain a branch named “development” where all features will be merged when completed
- Have a “production” branch where the final project version will be merged for our testing
- Recommended - Bitbucket or Github
- Provide means so we can access the latest data on the repository
- Upload all other required files to the repository, or include links to your private storage (Google Drive or otherwise) where you stored the required files

Use case example

This is a possible test scenario of the player’s actions:

1. The player moves in 2D XY space
2. The player picks up items
3. items go to the inventory (grid) and equipment slots
4. The player opens the inventory
5. The player equips the items from the grid to the equip screen
6. The player unequips the items from the equip to the grid screen
7. The player drops items to the ground
8. The player closes the inventory

FAQ

1. Inventory, Equipment and Attribute are separate windows
2. Player’s movement can be controlled while any, all or no windows from 1. are opened
3. Windows from 1 should not overlap, and each can be opened and interacted with separately

4. You can add separate sorting options to inventory if you want to, but there should be no auto-sorting, player must be able to move the item to any free slot in the inventory
5. The entire stack of an item can be moved at once
6. You define what you consider to be a single “feature” from the list of tasks to be implemented on a separate branch on the repository
7. You can find and use any applicable character sprite sheet on the Internet - or you can use this one.