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

Unity instructions:

- Use Unity - any version between 5.5.2f1 and 2018.2.0f2

Notes

- there is no need to waste time drawing your own graphics, you can download all the required graphics online, without any penalties
- Advice: check the accessibility, functionality and testability of all the data you send from another computer (builds, input touch, means to access the repository)
- All platform builds will be tested on 4:3 to 16:9 screen aspect ratios

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.apk** Android build

- **InventorySystem_WebGL.zip** containing WebGL build
- **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.
- If you solved the 13. Task, which is a bonus, provide a link to the repository (and access credentials if necessary) in the InventorySystem_README.txt

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. Bonus - 4 (orthogonal)/8 (orthogonal + diagonal) directional sprite sheet animation

1.3. Bonus – animation speed matches the movement speed

Link to sprite sheet character generation

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 at least one from the following options:

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

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

2.3. Option 3: Pick up items by physics trigger collision - on input

2.4. Option 4: Pick up items by player position + physics overlap circle - on input

2.5. Option 5: Pick up items by player position + physics overlap circle - continuous

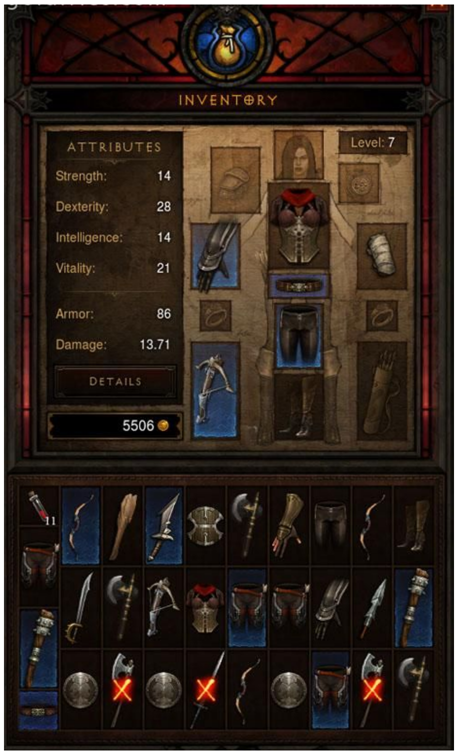
2.6. Option 6: Pick up items by physics circle casting in movement direction - on input

2.7. Option 7: Pick up items by physics circle casting in movement direction - continuous

2.8. 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.

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 are placed in the 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

6.4. Bonus: Stackable items

6.4.1. Upon item pickup, the item type/name is searched upon and either a new item/stack is created, or the number of items in the existing item/stack is increased.

6.4.2. Stackable item must have a display of item count in that cell grid (example: number of items in the stack in the lower left corner of the cell grid)

6.4.3. Bonus: display a maximum number of items in the stack. Vary maximum number in item types.

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

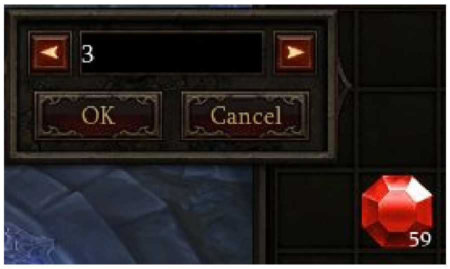
7.1.1. If the item is “in the air” and left mouse click is executed over inventory or equipment window, if the slot is of the appropriate type (equipment slot type), or it is an inventory slot, the item is slotted in the new cell

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. Stackable items can be split into 2 stacks of varying amounts by the split screen that appears upon interacting with the grid cell containing the stackable item. The window contains the left and right button, editable text field and buttons "ok" and "cancel"

7.4.1. Split window contains a slider.



3. Stackable item split screen example image – Diablo 3

7.5. 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.

7.6. 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.

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

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

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

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

7.11. Middle mouse button click on the consumable item consumes the item.

7.12. 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-equipable 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 UI button.

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

8.4. IMPORTANT: Item data must be inside a serializable class, 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

8.6. Bonus: Items have durability (current and maximum) where current durability is spent only on equipped items per x units of the Player’s walking distance. When the durability of an item expires, the item is unequipped from the slot automatically into the inventory.

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. Spendable attributes and Buff system

10.1. Player has 2 more attributes, health and mana, which are considered spendable attributes, unlike attributes listed in assignment 9.1., and as such, health and mana have maximum and current value, where the current value can be spent.

10.2. Items, when equipped, change the maximum or current value of attributes, by amount or percentage

10.3. Consumable items, when used, change attribute values in the following manner:

10.3.1. Hold bonus value over time (example: +30% strength over 4 seconds)

10.3.2. Ramp value up and/or down over time (example: change dexterity from 0 to +25 in 2 seconds, and hold +25 for 5 seconds)

10.3.3. Change value over time in tick manner (example: Damage Health by 20 over 3 seconds / Heal health for 3 per second over 6 seconds)

10.4. Create 1 Consumable item for each example change type, include them in spawning so they can be applied on the Player

10.5. Bonus: Display active buff duration in UI as Filled Radial360 image, support multiple active buffs display simultaneously.

11. Mobile touch input

Implement your own or add an external plugin for mobile touch input of your own choice, and implement the following options in the same unity project

Gestures - reactions

11.1. Long press - moves the item “into the air”

11.1.2 - on long press stop (finger release), if item was above empty inventory slot, it is slotted into that slot, if item was above the world space (there is no UI under the release position), item is dropped into the world, otherwise item is returned to the slot from which it was taken "into the air"

11.2. Touch up - releases the item from “the air”

11.3. Double tap - on an item equips/unequips equipable item / uses usable, consumable item

11.4. Pinch gesture - zooms the camera in and out between zoom limit values

11.5. Bonus: Pan gesture - if started over an item that can be split, and if ended over an empty slot or slot containing the same type of item - opens the splitting screen

11.6. Tap gesture - on the item in the inventory - shows the item tooltip

11.7. Implement one of the player character pick-up options so they can be picked up in the mobile build of the project, if necessary and if you choose so, you may add another UI button for “pick-up” interaction

11.8. Movement - choose one of the following:

11.8.1. Tap gesture - in the world - moves the character to the pressed world space location - choose one of the following

11.8.1.1. Movement is linear, from the current player position to the destination, and stops at colliders

11.8.1.2. Movement uses path-finding to reach the destination

11.8.2. Implement a mobile thumbstick/joystick input which will serve as horizontal/vertical input

11.9. Bonus: if you chose 11.8.1., prevent movement by tap if tap happens over the UI

12. Analytics

12.1. Implement in-game analytics with a plugin of your choice (Unity Analytics is a valid solution if you consider it’s suitable for all the requirements) for PC and mobile build.

12.2. Bonus: Implement in-game analytics for WebGL build.

12.3. Implement analytics events for the following:

12.3.1. On each build startup (platform, local time)

12.3.2. On Item Picked up (item name, item type)

12.3.3. On Item Equipped (item name, item slot)

12.3.4. On item Used/Consumed (item name, item type)

12.3.5. On window opened (window type)

12.3.6. On Player moved each 10 unity units total, with a maximum of 5 events for a single build execution.

12.3.7. On Player collided with the in-scene collider, with a timeout of 5 seconds minimum between 2 analytics sending events.

13. Bonus: Subversioning Development

- this is a bonus assignment, if you complete it, you do not need to link/send the required files in the response e-mail
- Create an 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 splits the stackable item and drops the split part of the stack
9. The player closes the inventory

Notes

- there is no need to waste time drawing your own inventory items/character
- you can download all the required graphics online without any penalties
- The executable file will be tested on 4:3 and 16:9 aspect ratio, 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, input touch, means to access analytics and repository)