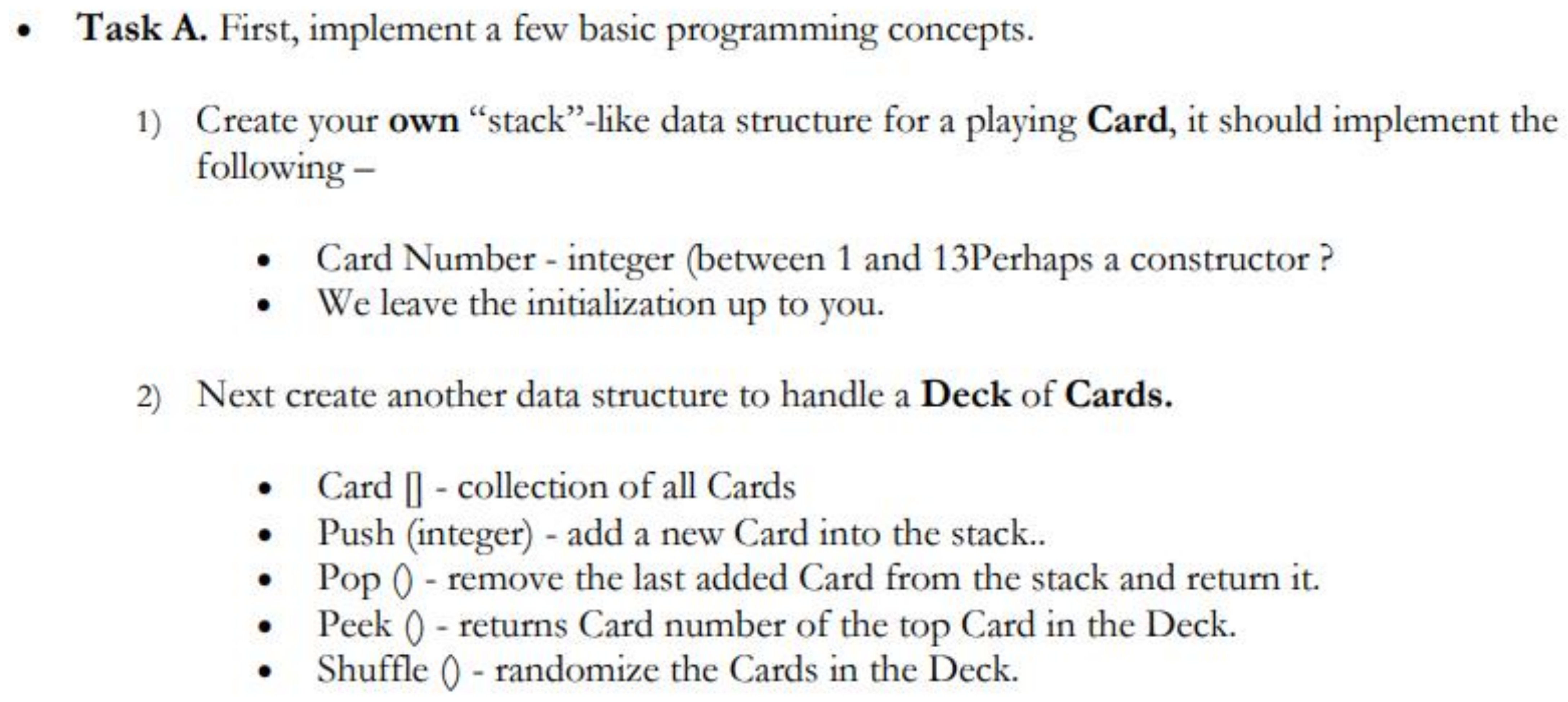
DOCUMENTATION

“This Documentation is created for having a detailed exploration of the entire design and development pipeline”

Content

- Understanding problem statement.
- Constructing essential scripting components.
- Creating Programmable Materials.
- Creating assets for game.
- Project settings and optimization

Understanding problem statement.



- **Task A.** First, implement a few basic programming concepts.

  1) Create your **own** "stack"-like data structure for a playing **Card**, it should implement the following –

     - Card Number - integer (between 1 and 13Perhaps a constructor ?
     - We leave the initialization up to you.

  2) Next create another data structure to handle a **Deck** of **Cards.**

     - Card [] - collection of all Cards
     - Push (integer) - add a new Card into the stack..
     - Pop () - remove the last added Card from the stack and return it.
     - Peek () - returns Card number of the top Card in the Deck.
     - Shuffle () - randomize the Cards in the Deck.

What is stack?

A stack is a linear data structure with LIFO architecture where user can only push at the top of the stack .

Traversal time -:Worst case scenario **time complexity O(N).**

Accessing an elements from the top of the stack **time complexity O(1)** .

Deletion of an element from top **time complexity O(1)** .

A stack is a **non primitive data structure** which can be either implemented by an **array or a linked list**.

When to choose array for stack implementation?

If we **know the size** of the elements we will be storing in a stack , it is best practice to store the elements in an array as it takes **contagious memory** , hence can be **randomly accessed using index**.

When to use a Linked list for stack implementation?

If we **do not know the amount of elements** we would be getting and the amount can **scale dynamically**

We will use a linked list , how ever random access is lost in case of linked list and it takes extra memory overhead for storing reference to its adjacent node in memory.

What are some common operations on stack?

Push (Inserting a new element at the top), Pop(Deleting an element from the top).

Why use a stack?

In our problem we need to build a deck of card where every **other card is stacked upon each other**.

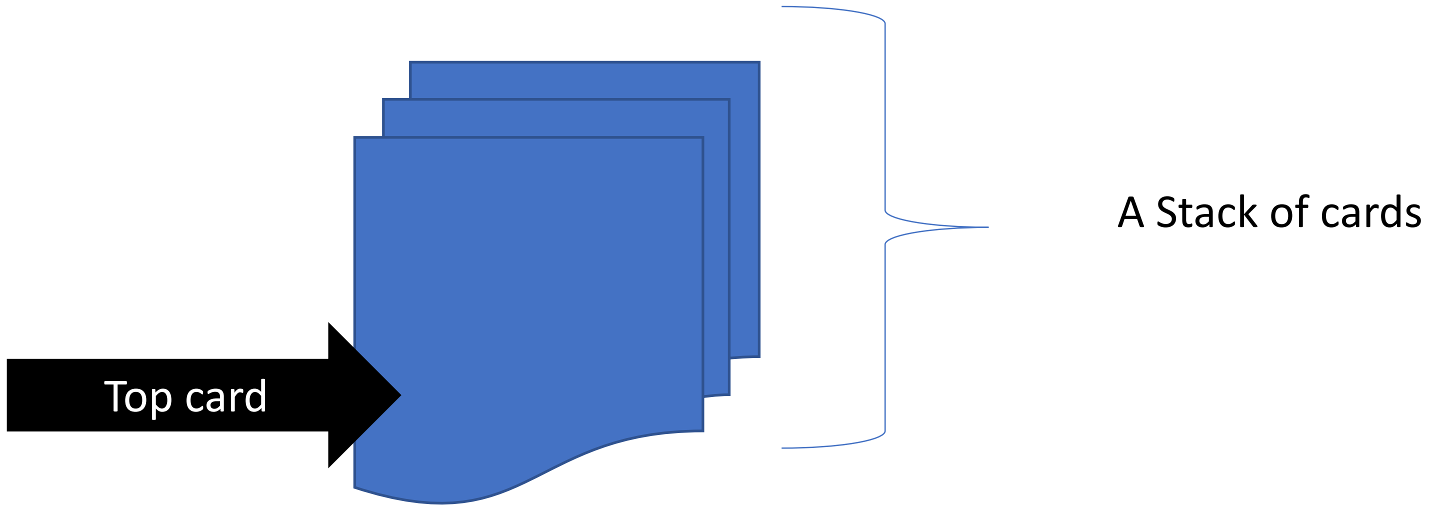
And we can only remove , insert , peek from the top of the deck . Hence **stack is the best suited** data structure.

What is the requirement?

Our requirement is to create a stack to store our cards which can range for 1 to 13 in face value , I have used

Poker Cards which ranges from **Ace to King** ,we need the ability to **push non duplicate randomly generated** numerical values which represents the cards .We also need the ability to **discard** a card from top of the deck.

We need the ability to **peek the top** card , and we also need the ability **to shuffle** the cards .



How did I approach the problem ?

I have created a **tokenizing** system for the game objects where they will be tokenized , there name will be assigned by the **numerical face value** of the texture.

This helps us to create a **hashset and check for duplication** , as we do not control the instantiation of the number directly and it is randomized.

I have **cached** the textures in a **dictionary** which can be accessed by the random number it self , we cast the randomly generated numbers to **strings key** , using **Enum**.

This string then helps us to fetch the texture.

Why string as the key ?

* Hence the question comes , would not it be easier to just use an integer for the key for dictionary , the reasons are many like it can be duplicate , it can be **hard to generate** as we will have to generate sequentially for each texture , and as we are **dynamically caching the textures** it would be hard to keep track of. Hence we are using the **Name of the texture file as key** it self.

This helps us to do all related operations with ease.

* For **pushing** we used a Top pointer which points to the top game object reference index in the Cards array.

The index increments when ever we push a element and we did not have all cards in our deck and also we check the hashset for duplication as we do not want same card to repeat more than once.

For **poping** we do use the same **global top variable** and **decrement the value** of the top pointer after discarding the top element . We also make sure to remove the element index from the **hashset** as **we do not want to cause any collision of values** while readding the cards , for this purpose we convert the **name of the card assigned to the game object.**

For **peeking** the top card again we use the same **Top pointer** , and access the game object and **rotate it 180** degrees as the card **will be facing to wards the user** and the **previous rotation was not facing** the user .

For **shuffling** we wrote a **coroutine** which shuffles the cards **over several frames ,** for **shuffling** we create two **random indexes** each ranging from **0-min(top+1 , 13)** as the **max range should be one grater** than the top or maximum value. Then we **interpolate the position** of the cards and **also swap the index** of cards after shuffling.

Constructing scripting components

We have created classes keeping in mind the individualization of responsibilities .

We have 4 different scripting components

- **Game manager**
- **Resource manager**
- **UI manager**
- **Card deck class**

Game manager-: Game manager class **initializes and controls** objects of all scripting components , hence **pipelining global resources** to all classes which needs them.

Resource manager-: Resource manager **caches all the resources dynamically** which are needed for the game .

UI manager-:UI manager handels all the **UI events** which gets called by **Event manager** e.g. button press .

Card deck class-: Card deck class is the most important class of our code base , as it defines the **stack and related functionality for manipulation**.

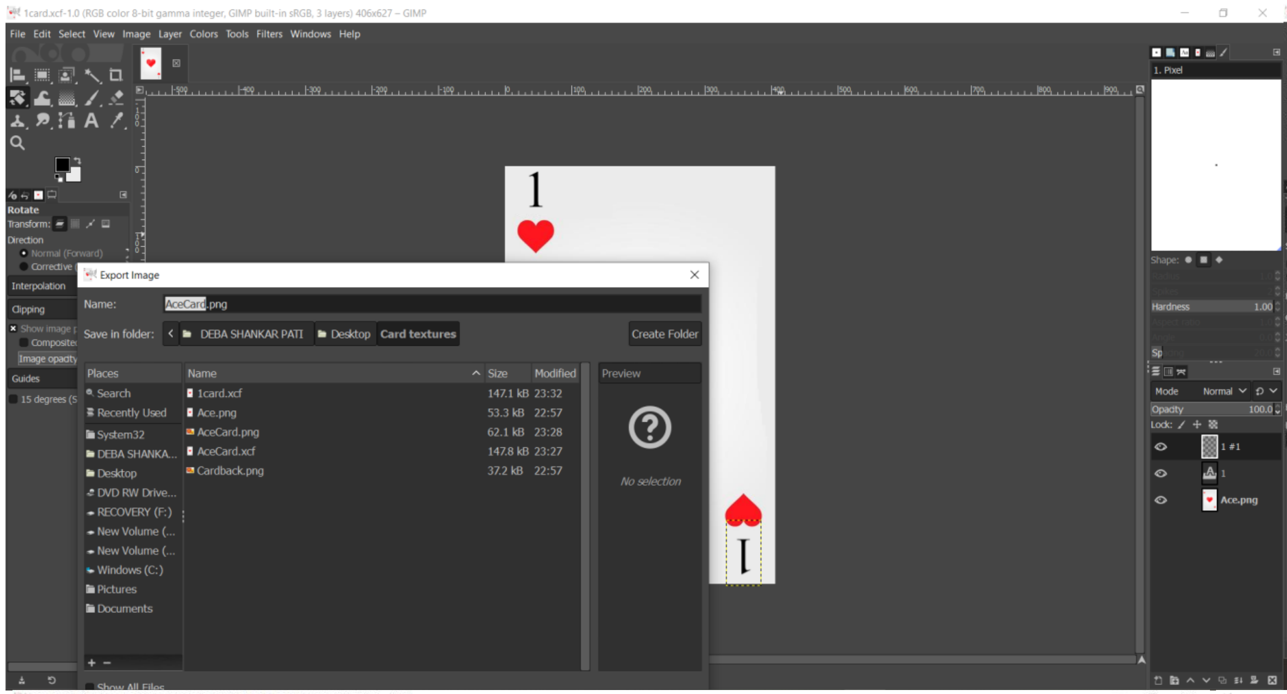
Creating Programmable Materials

We have written an unlit shader for our front face and a surface shader for the back face of the card planes.

The front face mesh displays the **face value (A-K)(Ace to king)** , the textures are fetched from the dictionary using the **random index and enumeration** , and **dynamically get added when we instantiate and push** card **prefab** to the stack.

For the back face we have a **common texture** and a **glowing outline** texture for differentiating the cards when stacked.

Creating assets for game

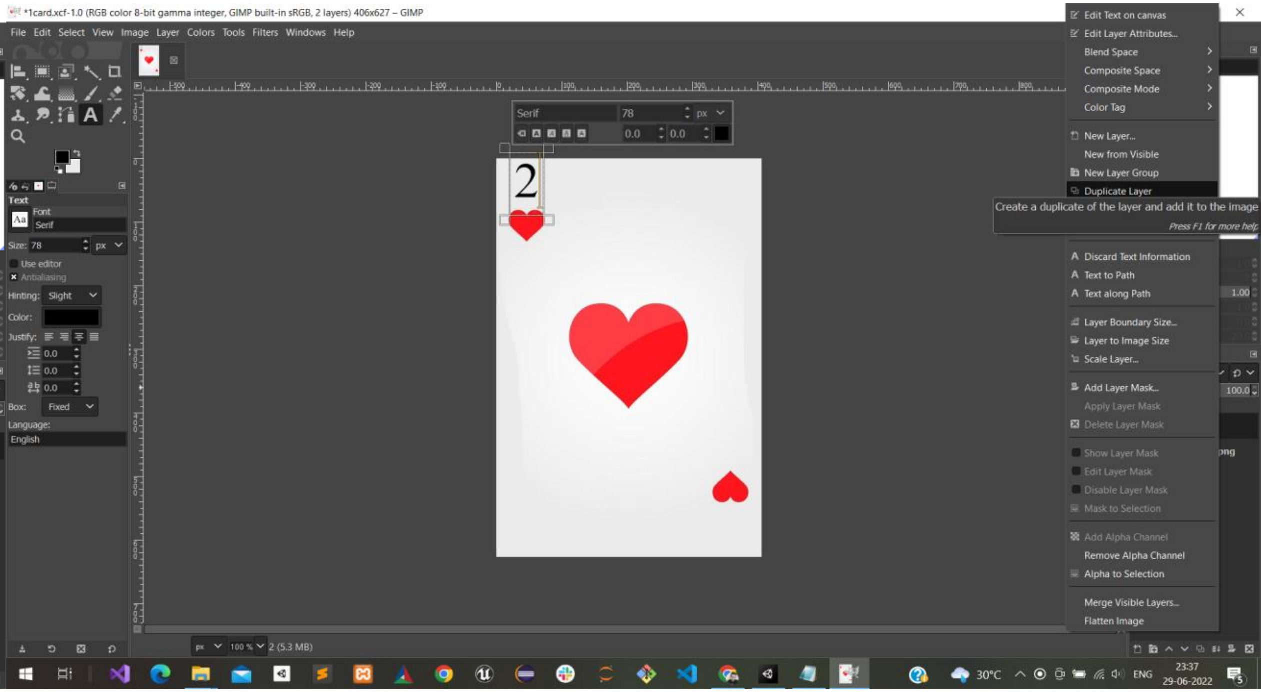

1card.xcf-1.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 3 layers) 406x627 – GIMP

Export Image

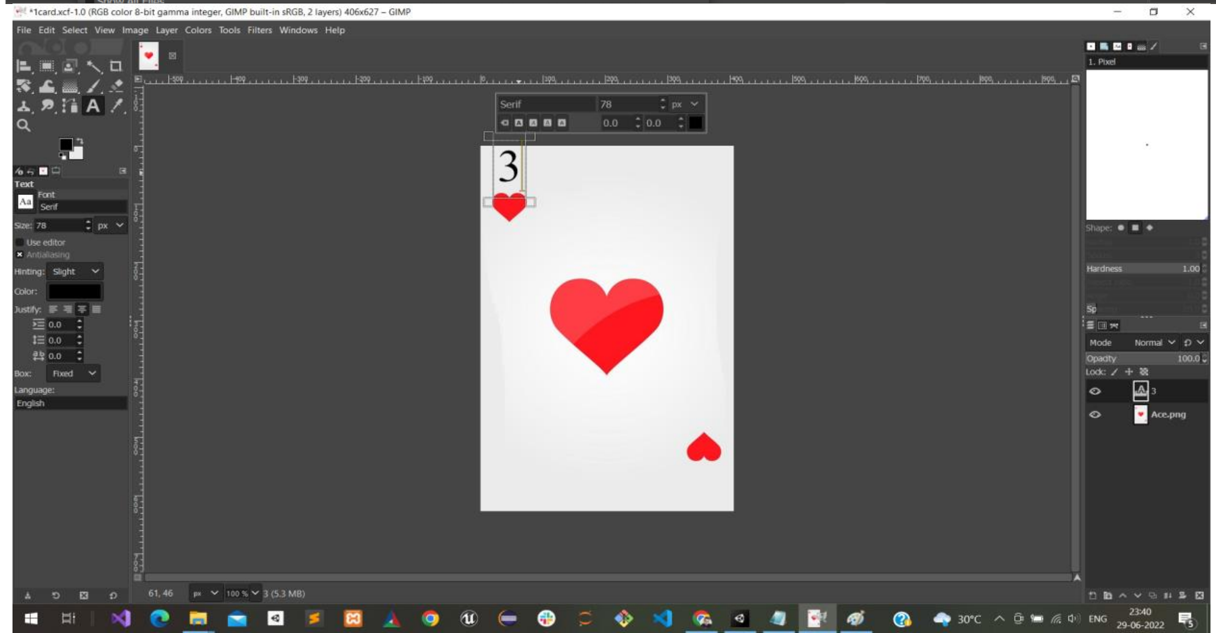
Name: AceCard.png

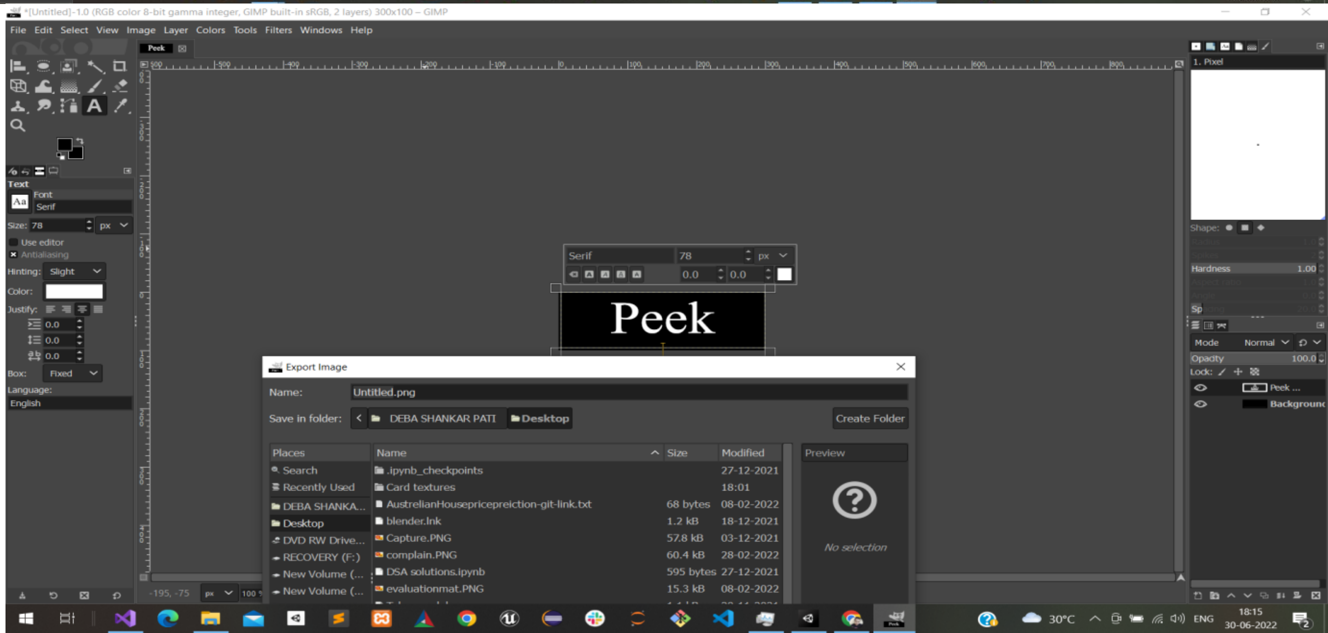
Save in folder: DEBA SHANKAR PATI / Desktop / Card textures

Name	Size	Modified
1card.xcf	147.1 kB	23:32
Ace.png	53.3 kB	22:57
AceCard.png	62.1 kB	23:28
AceCard.xcf	147.8 kB	23:27
Cardback.png	37.2 kB	22:57


*1card.xcf-1.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 2 layers) 406x627 – GIMP

Duplicate Layer: Create a duplicate of the layer and add it to the image


*1card.xcf-1.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 2 layers) 406x627 – GIMP

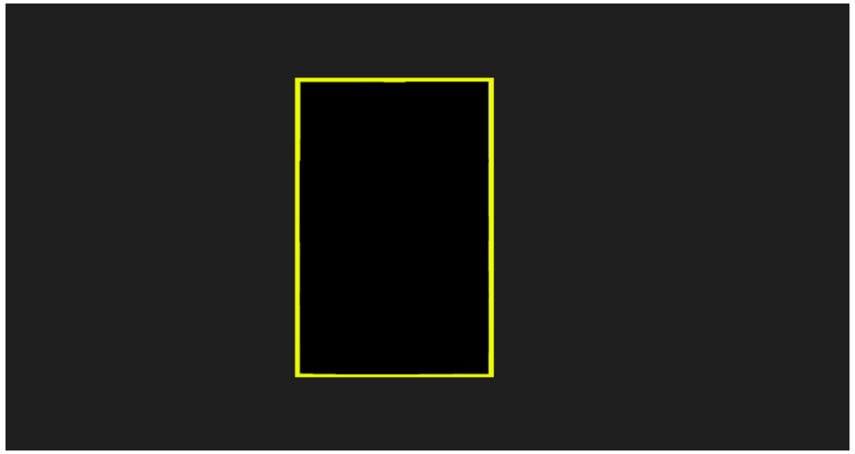

*[Untitled]-1.0 (RGB color 8-bit gamma integer, GIMP built-in sRGB, 2 layers) 300x100 – GIMP

Export Image

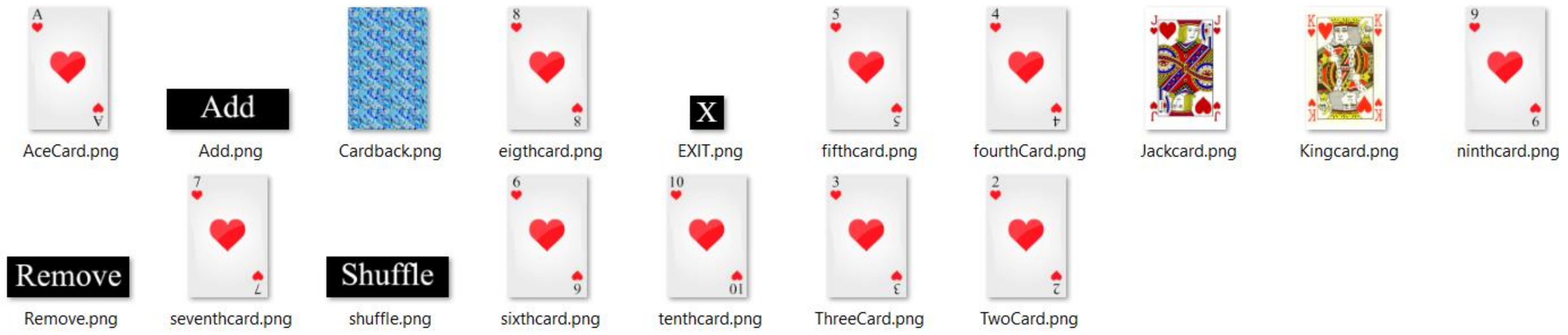
Name: Untitled.png

Save in folder: DEBA SHANKAR PATI / Desktop

Name	Size	Modified
.ipynb_checkpoints		27-12-2021
Card textures		18:01
AustrelianHousepricepreiction-git-link.txt	68 bytes	08-02-2022
blender.lnk	1.2 kB	18-12-2021
Capture.PNG	57.8 kB	03-12-2021
complain.PNG	60.4 kB	28-02-2022
DSA solutions.ipynb	595 bytes	27-12-2021
evaluationmat.PNG	15.3 kB	08-02-2022

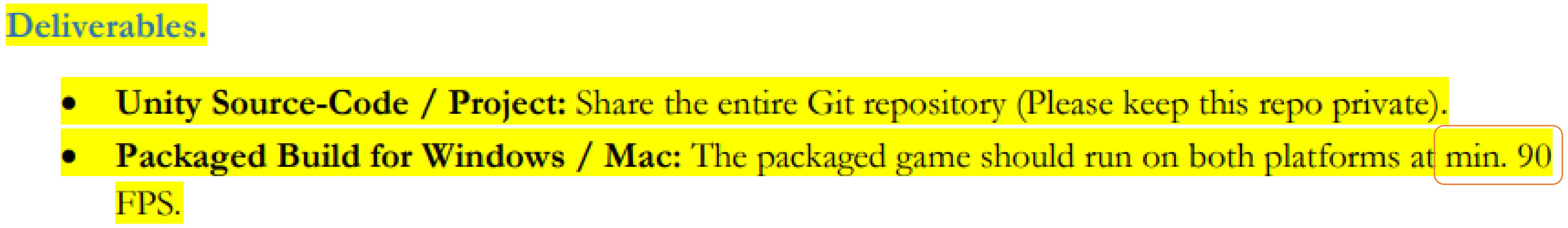


All textures and sprites are created In GIMP



Project settings and optimization

- We disabled the shadow and reflection probe as we do not need in our game.
- We have enabled GPU instancing for materials which helps us to batch meshes and reduce render calls.
- We have removed mesh colliders as we do not need any physics.
- We disabled Vsync as this was limiting our frame rate to display refresh rate , in this case we do not need that as we are not using any movement or physics in our game , we do not have to worry much about screen tearing. And our requirement states to have at least 90 FPS , which was not possible as my display refresh rate was lesser than 90 FPS .



Deliverables.

- **Unity Source-Code / Project:** Share the entire Git repository (Please keep this repo private).
- **Packaged Build for Windows / Mac:** The packaged game should run on both platforms at min. 90 FPS.