# ENGR 1181 | Software Design Project

# Project Description

# Notebook Guidelines

# Extra Credit Opportunities

# Simple Game Engine Documentation

# References

# Project Description

## Learning Objectives:

This software design project challenges students to combine problem-solving strategies, the design process, and programming skills in a team setting to solve a real-world programming scenario. Students will review this document before arriving to lab.

## 1. Problem Statement

Each team is tasked with developing a game (or games) using MATLAB and documenting the project. The game(s) you choose should be relatively simple; good choices include implementing dice, card, and board games in MATLAB, but you are also free to make up your own game. Regardless if you use an existing game or make up your own, be sure to have a good understanding the rules for it before you sit down to program your game, since you will have to implement those rules in MATLAB. A good resource for playing some dice, card, and board games can be found here: [1].

Since different games can vary on their difficulty to create, each game will be assigned a complexity score from 1-4. Some examples of these can be seen in Table 1 for reference, but you should not let those options hamstring you: if you have an idea for a different game, ask your instructor or GTA to provide you a Complexity Score (CS) for it. Each team must have a total complexity score (TCS) of at least 2 working games for full credit on the project (more for extra credit, see below). Additionally, all members of the team must contribute to coding the games. This does not need to be equal, but each team member must contribute to the coding of about 0.5 CS.

To aid in the development of the games, we have developed a simple “game engine” to make creating visualizations and input easier. The documentation for this is provided below. We encourage you to use this system, but it is not mandatory. As with any other reference you use, you MUST reference the material in your final documentation.

Table 1: Example complexity scores for MATLAB games [2]

|  |  |
| --- | --- |
| **Complexity Score (CS)** | **Games** |
| 1 | Over/Under Seven, Craps, Simple dice games |
| 2 | Memory, Crazy Eights, Go Fish, Hangman, Mancala, Can’t Stop |
| 3 | Battleship, Connect 4, SOS, Blackjack, Mastermind, Lingo |
| 4 | Battleship (smart AI), Connect 4 (smart AI), Yahtzee, Othello |
|  |  |
| 2-4 | Adventure or role-playing game, depending on graphics (text only is okay), game mechanics, and size |
|  |  |
| +1 (max 4) | Smart computer player (where applicable) |

## 2. Project Tasks – See Carmen for Exact Due Dates

* SDP 1: Brainstorming/Planning Day
  + Create a u.osu.edu Account – Due SDP 1 End of Class (EOC)
  + Initial Decision on Game/Games – Due SDP 1 EOC
  + User Interview #1 (u.osu.edu) – Complete SDP 1 EOC, Upload SDP Day 2
* Before SDP 2
  + Follow Instructions to Create u.osu.edu site and import template – Due SDP 2
  + Create or Update Team Working Agreement (u.osu.edu) – Due SDP 2
  + Individual Responsibility Agreement (u.osu.edu) – Due SDP 2
  + Flowchart, Algorithm, or Pseudocode Draft (CarmenCanvas) - Due SDP 2
* SDP 2: Work Day
* SDP 3: Work Day and Beta Testing
  + User Interview #2 (u.osu.edu) – Due SDP 3 End of Class (EOC)
  + Final Software Design Project Plan – Due SDP 3 End of Class (EOC)
* SDP 4: Work Day and Final Testing (Must have final test by instructional team by end of class)
* Documentation and Video
  + Draft Video (CarmenCanvas) – Due SDP3
  + Pitch Video (u.osu.edu) – Due 1 day before Class 25
  + Final Documentation (u.osu.edu) - Due Class 25

# Project Notebook Guidelines

## Project Notebook Description:

Each team will complete a project notebook on a u.osu.edu site. This notebook is due during Class 27. The notebook should follow the guidelines given below. Teams should keep in mind that this notebook should be well organized and professional in appearance.

## Project Notebook Requirements:

The project notebook should be organized as follows:

1. Welcome Page
   1. Table of contents describing what can be found in each link
   2. Executive Summary
2. Project Management Documentation:
   1. Team working agreement
   2. Individual Responsibility agreement
   3. Project schedule
   4. Meeting notes
3. Business Plan:
   1. User Identification and Interviews – Identify the intended audience for your game, conduct 1 interview with a potential user PRIOR to development, conduct 1 interview with a user after beta testing. The questions for these interviews can be found on your u.osu.edu website once you have applied the template.
   2. Electronic/Print advertisement for the game – (flyer, brochure etc.)
   3. 30 second to 1 minute pitch video/advertisement for your game(s). All team members must speak and/or appear on the video.
4. Software Documentation
   1. Introduction
   2. User Manual
      1. This description is for a general audience. This audience may not be familiar with programming. A reader with no MATLAB or engineering knowledge should be able to understand this description.
      2. This section should describe all aspects of the program.
   3. Program Description for Developers
   4. This is a technical description of your program. The intended audience includes those familiar with MATLAB, the provided MATLAB game commands, and programming in general.
   5. Include a list of variable names and uses. This should include **every** variable that is used within your code.
   6. Include a list of the provided MATLAB game commands used with short descriptions. Do not copy the descriptions from the provided documentation.
   7. Students are not permitted to copy any project descriptions from this document or any provided documentation.
   8. You must reference any outside sources used in the development of your game. Include in Reference section as well.
   9. Final Algorithm, Flowchart or Pseudocode/Flowchart
   10. Write out an algorithm, flowchart or pseudocode to match the team’s final program.
   11. This may be similar or drastically different from the team’s initial draft based on how much the code changes over the course of the project.
   12. Final Program with Comments
5. Include the code from the final program used during testing.
6. The code MUST have sufficient comments throughout. Another programmer should be able to read these comments and have a good understanding of how the code works.
   1. Brief Discussion
7. Provide brief explanations of what occurred during the game’s testing.
8. Describe the progression of the MATLAB code.
9. Describe the obstacles faced and how they were overcome.
   1. Conclusion and Recommendations
10. Develop a conclusion from the results obtained throughout the game software design project, including the game itself, the programming, and problem solving method. This can also include recommendations for changes to the game if you had more time for development or if you were going full scale with the product.

# Extra Credit Opportunities

Extra credit is available if you exceed the maximum number of game points. This can be done through more difficult games and/or additional games. You may earn the extra credit for creating the additional games or for creating the additional games and creating additional documentation. You must complete Extra Credit Opportunity #1 to be eligible to complete Extra Credit Opportunity #2.

Extra Credit Opportunity #1

Exceed the 2 TCS minimum – Extra Credit earned in Demonstration/Testing Assignment

|  |  |  |
| --- | --- | --- |
| 3 TCS of games demonstrated/tested |  | 5 points of extra credit |
| 4 TCS of games demonstrated/tested | Must include a game with CS of at least 2 | 10 points of extra credit |
| 6 TCS of games demonstrated/tested | Must include a game with CS of at least 3 | 15 points of extra credit |

Extra Credit Opportunity #2

Exceed the 2 TCS minimum in the Notebook Documentation – EC earned in Notebook Assignment

|  |  |  |
| --- | --- | --- |
| 3 TCS of games documented |  | 5 points of extra credit |
| 4 TCS of games documented | Must include a game with CS of at least 2 | 10 points of extra credit |
| 6 TCS of games documented | Must include a game with CS of at least 3 | 15 points of extra credit |

# Simple Game Engine Documentation

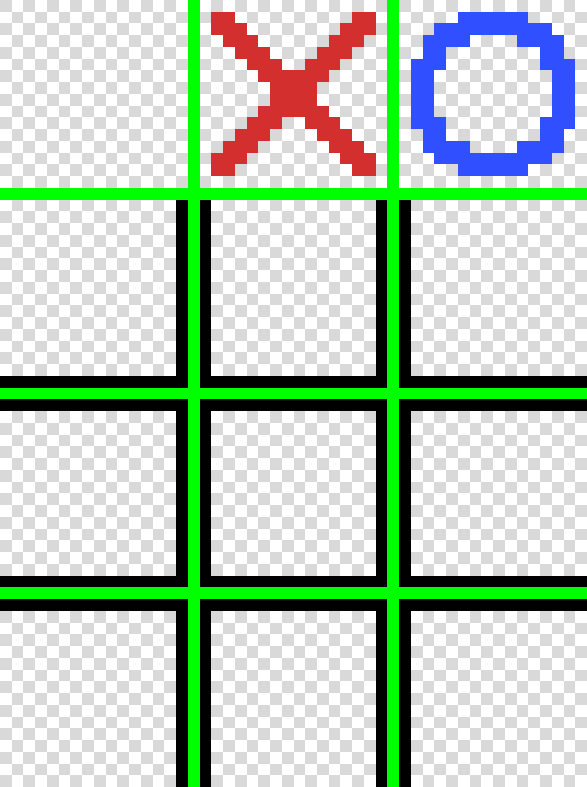
The Simple Game Engine (SGE) is a collection of functions [3] to make developing a game in MATLAB easier. The download can be found on Carmen in a single m-file: “simpleGameEngine.m”.

The basic idea is to imitate (a much simpler version of) how real 2D games are made [4]. The graphics for each movable element of the game (characters, items, etc) are called *sprites*. The sprites are pre-drawn and saved to an image file called a *sprite sheet*. The engine loads in this sprite sheet, and layers the composite sprites onto a *background* and onto each other to create a *scene*. The game code is what determines which sprites go where and when.

In the case of the SGE, the background is a single color, and the sprites are arranged on a grid with a maximum of two layers. There are also functions to get keyboard and mouse input from the player.

## Sprite Sheet

The sprite sheet is a PNG image with the sprites for the game laid out in a grid with a one pixel wide buffer between them. PNG images are used because they can support transparency. Each sprite needs to be the same size, but there is no restriction on what this size is, and rectangular (i.e. not only square) sprites are allowed. How you arrange your sprites in the sprite sheet will determine their identification number (or sprite ID) used in the SGE; the sprite at the top left of the image is #1, then the sprites are numbered sequentially left-to-right and top-to-bottom. See the example sprite sheet for Tic-Tac-Toe in Figure 1.



Sprite 1 (empty)

Sprite 2

Sprite 9

Checkerboard pattern indicates transparency in most image editing programs

Green lines denote the buffer pixels to help align your sprites and will not show up in game

**Figure 1:** Example sprite sheet for Tic-Tac-Toe. Enlarged and annotated.

Some example sprite sheets (as well as a blank one) are provided for you (all with 16-by-16 pixel sprites), which you can edit (using a pixel graphics editor [5] makes things much easier) or use as-is. These sprite sheets are based on the 1-Bit Pack found here: [6], which is also a good reference for additional free sprite sheets.

## Initialize the Scene

A new type of variable is introduced by the SGE: a scene variable [7]. These are special variables that store the sprite sheet and figure information for your game. Before you can draw any scenes, you need to tell MATLAB about the sprite sheet and background color that you want. To do that, you use the initialization function named simpleGameEngine.

**Function**: simpleGameEngine

**Input**:

1. File name of sprite sheet as a character array
2. Height of the sprites in pixels
3. Width of the sprites in pixels
4. (Optional) Zoom factor to multiply image by in final figure (Default: 1)
5. (Optional) Background color in RGB format as a 3 element vector (Default: [0,0,0] i.e. black)

**Output**: an SGE scene variable

**Note**: In RGB format, colors are specified as a mixture of red, green, and blue on a scale of 0 to 255. [0,0,0] is black, [255,255,255] is white, [255,0,0] is red, etc.

**Example**:

my\_scene = simpleGameEngine('tictactoe.png',16,16,5,[0,150,0]);

This function doesn’t actually make a new figure, it just sets up the engine.

## Draw the Scene

To draw something to your new scene use the drawScene function. The basic idea is you just need to specify what sprite to draw where (and in which layer). To do this you need to somehow create a matrix (or two matrices if you want to use both layers) with the ids of the sprites in them. The engine will then create (or update) a figure with those sprites drawn in those locations.

**Function**: drawScene

**Input**:

1. an SGE scene, which gains focus
2. A matrix of sprite IDs, the arrangement of the sprites in the figure will be the same as in this matrix
3. (Optional) A second matrix of sprite IDs of the same size as the first. These sprites will be layered on top of the first set.

**Output**: None

**Note**: Although nothing is returned, like the plot function, a new figure will be created (or the old figure updated).

**Example**: The following will create a figure with 3 rows and 3 columns of sprites

drawScene(my\_scene, [4,5,6;7,8,9;10,11,12], [1,1,1;1,2,1;1,1,1]);

## Keyboard Control

One option for allowing the user to control your game is to use the keyboard as the controller. It is very common in games that require movement to use the arrow keys, or to use keys on the keyboard as shortcuts for various actions. SGE has a function named getKeyboardInput to facilitate this.

**Function**: getKeyboardInput

**Input**: an SGE scene, which gains focus

**Output**: next key pressed while scene has focus

**Note**: the operation of the program pauses while it waits for input

**Example**:

k = getKeyboardInput(my\_scene);

Below is a list of the values returned based on different key presses.

|  |  |
| --- | --- |
| **Keyboard Key** | **Value Returned** |
| Up arrow (↑) | 'uparrow' |
| Down arrow (↓) | 'downarrow' |
| Left arrow (←) | 'leftarrow' |
| Right arrow (→) | 'rightarrow' |
| Enter | 'return' |
| Spacebar | 'space' |
| Escape | 'escape' |
| Backspace | 'backspace' |
| Numbers along the top (e.g. 1) | '1' – character of the number pressed |
| Numbers on the number pad (e.g. 7) | 'numpad7' |
| Letters (e.g. a) | 'a' – character of the letter pressed |

## Mouse Control

Another option for allowing the user to control your game is to use the mouse and have the user click on different parts of your scene to interact with the game (for example, to choose which cards to keep in draw poker). You can do this with the getMouseInput function.

**Function**: getMouseInput

**Input**: an SGE scene, which gains focus

**Output**:

1. The row of the tile clicked by the user
2. The column of the tile clicked by the user
3. (Optional) the button of the mouse used to click (1,2, or 3 for left, middle, and right, respectively)

**Notes**: A set of “crosshairs” appear in the scene’s figure, and the program will pause until the user clicks on the figure. It is possible to click outside the area of the scene, in which case, the closest row and/or column is returned.

**Example**:

[row,col,button] = getMouseInput (my\_scene);

## Going Further

The Simple Game Engine is essentially a wrapper for a few built-in MATLAB functions: imread, imshow, waitforbuttonpress, get, and ginput. If you would like to expand the functionality of the SGE, you are encouraged to read the MATLAB documentation on these functions and look through simpleGameEngine.m (it may help to learn a bit about classes and object-oriented programming). The simplest modification you could make would be to add an additional layer to drawScene. Also, your book has additional unassigned chapters that you may find useful.

Some additional functionality to look into:

* Sound (easy): read the documentation on audioread and sound
* Message and Error boxes (easy): read the documentation on msgbox and errordlg
* Add text to figures (easy): read the documentation on text
* Replace the background color with a background image (moderate): you will need to modify both simpleGameEngine and drawScene to accomplish this. Hint: look at how the sprite sheet is loaded and handled and do something similar.
* Real-time game execution (difficult): have your main game loop execute 30 times per second using pause (as well as tic/toc). For user input, use guidata to pass what key was pressed back to your game. Try running the following code and see if you can figure out how it works:

framerate = 30; % frames per second

rtg\_scene = simpleGameEngine('retro\_simple\_dice.png',16,16,10,[0,0,0]);

drawScene(rtg\_scene,[8]) % draw the scene

while 1 % main game loop

tic

key\_down = guidata(rtg\_scene.my\_figure) % user input

% Use key\_down to determine a sprite ID

if key\_down

sprite = str2double(key\_down);

else

sprite = 0;

end

% if it is a valid sprite ID (between 1 and 6), display it

if sprite >= 1 && sprite <= 6

drawScene(rtg\_scene,[sprite])

else

drawScene(rtg\_scene,[8])

end

pause(1/framerate-toc); % wait for next frame

end

* Draw sprites anywhere, not just on a grid (difficult): modify drawScene to take vectors of sprite IDs as well as their x,y-coordinates, and use those to draw sprites on the figure.

# References

1. “Flash Board Games Online” Available at: [www.playboardgameonline.com](http://www.playboardgameonline.com) [Accessed 9 Jul. 2019].
2. Ossman, K., and Bucks, G., “First Year Student Team Projects Using MATLAB,” *First Year Engineering Experience Conference*, August 8-9, Pittsburgh, PA, 2013.
3. Actually, the Simple Game Engine is a “class” from object-oriented programming. These are a higher level of organizing a program beyond functions. Classes are collections of related data and functions (called methods). Much of MATLAB itself is organized this way.
4. Loew, A., “What is a sprite sheet?” Available at: <https://www.codeandweb.com/what-is-a-sprite-sheet> [Accessed 9 Jul. 2019].
5. A free online pixel editing program can be found here: <https://www.pixilart.com/draw> [Accessed 9 Jul. 2019].
6. “Kenney – 1-Bit Pack” Available at: <https://www.kenney.nl/assets/bit-pack> [Accessed 9 Jul. 2019].
7. This is the object of the Simple Game Engine class.