Snake: Technical

The program starts by outputting a string containing the controls to the consol. Then enables keyboard interrupts. The function runs sequentially into

Begin: which Initializes relevant global variable. Begin drops into

LevelStart: calls DrawArena, ScoreDisplay, and ClockDisplay to overwrite any possible artifacts on screen. It then creates a snake by initializing head_ptr, and tail_ptr and loading addresses for 3 consecutive location units (a location unit is a single number that corresponds to a x,y cord on the display given the formula (x/8)+((y/8)*32)) into the snake_body which is an implementation of a queue. It then draws the snake at it's given locations and calls AppleGen 5 to generate 5 apples. it then drops into

main: the main function for this program is an infinite loop that calls the GameClock which waits till 200ms has passed since last game cycle. It then updates the display which reflects the incremented number of cycles that have passed. It then read the Input to see if the user has pressed a valid key since last game cycle. It if the user has hit 'p' it calls Pause, if the user hit 'r' it calls Restart, if the user hit 'm' it calls Menue, if the user hit 'e' it calls Exit, else the main function calls the SnakeTracker before looping back to the top of main.

SnakeTracker: takes the value passed to it by main, containing the last valid direction input from the user, this value is interpreted and checked to make sure it is not against the current direction of travel, if it is it is discarded and replaced with the current direction of travel. the function then takes new valid direction or old direction if it was not valid and calculates the location of the next unit in that direction stored in the memory pointed to by the head_ptr. it passes this value the Collision Handler and uses the response from the collision handler to decide if the snake has hit an apple or not. if an apple is not hit the tail_ptr pops one value, if an apple was hit this step is skipped.

CollisionHandler: Calls ReadObject which returns the color of the object passed to it. The collision handler takes this information and determines if the snake hit an apple, wall, itself, nothing. if it hit itself or a wall the program immediately jumps to the GameOver function. if it hit nothing it passes back 0, if it hit and apple it calls AppleGen and ScoreKeeper to create a new apple and passes back a 1.

AppleGen: generates a random (x,y) value then calls ReadObject to check if an object is already at that location. if not it places an apple there, if there is it loops back to itself until it successfully places an apple at a valid location.

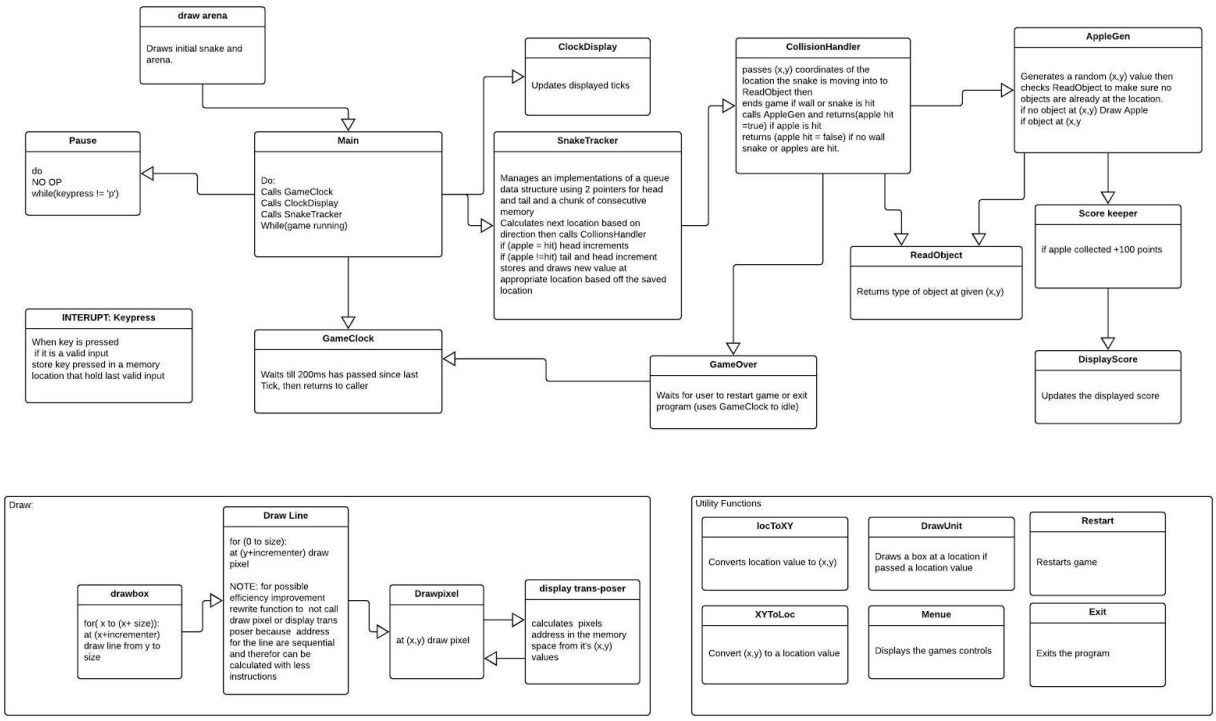
ScoreKeeper: adds 100 points to the score.

ClockDisplay and PointDisplay: both take their respective values remove the lowest significant bit by using div/10 and taking the remainder. they then us that value to call the OutText function which draws a number on the bitmap. they then take the quotient and repeat this to find the next least significant bits.

Pause: puts the game in an idle loops until the user hits 'p' again

GameClock: puts the game in and idle loops until 200ms has passed since last Tick, it then saves this time before returning so that it can measure from last tick allowing other game processing time to not slow down the games clock.

ReadObject: function similar to DrawPixel, but instead of writing to a memory location it reads from a memory location then does a reverse look up from the Color table to find out the color code for a pixel at a given location.



Build stages:

1. **New routines:** DrawArena, ReadObject
Existing routines: CalcAddress, GetColor, DrawDot, HorzLine, DrawBox, OutText
Routines testing: test code was added to to the ReadObject that output the object code every before it returns to the calling function, static test cases were drawn to the display and the function called to verify it returned the correct values.

2. **New routines:** LevelStart, SnakeTracker, LocToXY, XYToLoc, DrawUnit
Design changes: The interpretation location data was the game uses was changed to integer values between 0-1023 that are mapped to units of 8 by 8 pixel locations. this was down primarily so that location data could be saved as a value rather than separate x and y value. LocToXY, XYToLoc, DrawUnit were added to handle the new functionality. A possible future function to redraw the snake was considered that if implemented should be utilized to draw the initial snake.

Routine testing: LevelStart was run to verify would draw correctly. A simplified version of the controller was implemented to test the basic movement of the snake to validate the SnakeTracker function, and to further test the functions it utilizes. Each draw function was changed to a unique color to easier track. Some colors appeared to misaligned, however after adjusting magnification of the display it was determined this the problem was with the way mips draws pixel in the Bitmap Display and not a bug in the game.
Errors corrected: LocToXY contained a calculation bug where it failed to adjust the xy values by a factor of 8.

3. **New routines:** CollisionHandler, AppleGen,
Design changes: changed naming convention from CollisionTest to CollisionHandler to better reflect the functions role. Because of the order CollisionHandler and SnakeTracker were implemented, SnakeTracker now calls CollisionHandler to see if the snake hit something rather than testing before calling SnakeTracker.
Routine testing: Game was played in current state Initially using different colors for different draw calls to track different functionalities.
Errors corrected: SnakeTracker was checking checking for the container end using Snake_body rather than Snake_body_end, AppleGen was passing the wrong register when calling it's check to protect against placing apple on top of other objects.

4. **New routines:** GameClock, InteruptKeyPress,
Design changes: The GameClock implemented to start it's wait cycle between ticks before returning the processor to the game, to allow smoother game play, and to remove the case of inconsistent tick speeds in various cases. Timer name changed to GameClock. Speed changed to TickPeriod to bring it in line colloquial naming. SnakeTracker updated to prevent user from attempting to move opposite the last direction of movement. Game was tested at various speeds to find a TickPeriod that felt to play at, current speed is 200ms or 5ticks a second.
Routine testing: Many lines of temporary test code were removed to allow the game to be run as is. TickPeriod was adjusted to different values to verify it properly controlled the GameClock function. Outputs were temporarily added to Interput:KeyPress to validate it.
Errors corrected: GameClock was missing jr $r causing it to run into the pause function which was located below it. The new branch statement handling when the user attempted to move opposite the last direction of movement all pointed at one location causing the snake to move up one then successfully move the way the user was attempting, this was almost left in as a feature.

5. **New routines:** ScoreKeeper, ScoreDisplay, ClockDisplay
Design changes: The user no longer gains points for survival, currently all points given by apple collection, this was done as a more elegant solution to address the problem of a player avoiding apple. Because the OutText function we have only has char up to 'f' providing a menue in the bitmap display was scrapped because of the time required to build a full char list bitmap, User instructions and highscore display, and menue functionality has been moved to the console display. ScoreDisplay and ClockDisplay still display to bitmap display.

Routine testing: ScoreDisplay and ClockDisplay were tested with hard coded values. Score Keeper was tested while given different possible in game scenarios, and tested by calling it in a test function to verify it worked properly with given parameters. Game tested to score of 14300.
Errors corrected: There was a problem with the way OutText copied which was causing recurring problems with the program, unfortunately the problems are corrected, but the source of the problems are not. Because the way the game checks for walls the score righting over part of the wall made it no longer detect the walls under the score. The arena was moved down one unit to correct this.

6. **New routines:** Pause, Restart, Menue, Exit, GameOver
Design changes: Since no NULL case is being handled with the input catcher, all input driven function calls other than valid directions will set Input = Last_Direction after being read to simulate it being set to a NULL value. Menue currently is just a placeholder that displays the controls. HighScore disabled at this time due bug introduced to kernels by the custom exception handler. Automatically exiting after loss changed to an idle loop that waits for user to restart level or exit game. UpdateDisplay function call was moved from GameClock main functions inf loop so GameClock could be reused as an idle function in the GameOver loop.
Routine testing: features were implemented then tested in game.