Combat (Atari 2600)

The full game also involves air vehicles, but I only focus on the setup for tank levels. They distinguish themselves over the 14 levels by adding tank invisibility, bouncing bullets, longer bullet duration and different level layouts. I have chosen not to map the level layout in this written specification.

Main Program	**class Tank**	**class Bullet**	**class Wall**	**class Score**

// Variables
 	
Score score;
Tank player1;
Tank player2;
Bullet p1Bullet;
Bullet p2Bullet;
Wall[] walls;
 	
int level = 1;
lastTick = millis();
	// Variables
	
	int posX, posY, t_width, t_height;
	int thrustSpeedX, thrustSpeedY;
	int rotationAngle;
	
	boolean invisibility, dying;
		// Variables
		
		int posX, posY, b_width, b_height, velX, velY;
		int rotationAngle;
		
		boolean bounce;
		int duration, currentTick;
			// Variables
			
			int posX, posY, w_width, w_height;
				// Variables
				
				int player1 = 0;
				int player2 = 0;
				boolean player1Scored = false, player2Scored = false;
				
				txtPosX, txtPosY, txt_width, txt_height

// Functions
 	
void setup(){}
//set up screen size
//set up objects except bullet and walls and prepare level 1
 	
void keyPressed(){}
//see notes
 	
void buildLevel()
//see notes
{
if(level == 1) {}
else if(level == 2){}
else if(level == 3){}
}
 	
void draw(){}
//update deltaTime
//update all objects
//if running through updates, lastTick = millis();
}
	// Constructor
	Tank(){}
	//player1.rotationAngle is set to face right
	//player2.rotationAngle is set to face left
	//depending on buildLevel, invisibility is true or false
	
	// Methods
	void update(){}
	//if dying = true, run only dyingPhase() and display
	else
	//collision, move, rotate, shoot, display
	
	void collision(){}
	//updated before player actions. Check for collision with walls. See notes.
	
	void moveForward(){}
	//add thrustSpeedX and Y to player.posX and Y matching the direction of rotationAngle that player is driving
	
	void rotate(){}
	//see notes
	
	void shoot(){}
	//new Bullet();
	//tank spawns a bullet object with a tag or using variable name to indicate its owner
	//bullet also receives the current rotationAngle and position of shooting player
	
	void dyingPhase(){}
	//see notes
	
	void display(){}
	//if tank.invisibility = true, only display tank while bullet is shot and not expired
		// Constructor
		Bullet(int tankRotationAngle, int tankPosX, int tankPosY){
		bulletRotationAngle = tankRotationAngle;
		posX = tankPosX;
		posY = tankPosY;
		//depending on buildLevel bounce is true or false and duration may be 1.5 or 2.5 seconds
		}
		
		// Methods
		void update() {}
		//order: expire, collision, move, display
		
		void collision() {}
		//updated before actions.
		//player_.dying is true if hitting other player
		//see notes
		
		void move() {}
		//see notes
		
		void expire() {}
		//currentTick++;
		//if colliding with wall && bounce = false || currentTick of Bullet exceeds duration.
		Calling it first in update hinders bullet from being updated in collision, move and display
			// Constructor
			Wall(){}
			
			// Methods
			void update(){}
			//display
			
			void display(){}
				// Constructor
				Score(){}
				
				// Methods
				void update() {}
				//check if someone scored, if so updateScore
				
				void updateScore() {}
				// Either player1 or player2 increases by 1. Then Scored-boolean is set to false
				
				void display() {}
				//display above and outside playing field

Link to game: https://www.free80sarcade.com/atari2600_Combat.php

Description of game requirements:

- Screen resolution: 160x192 pixels
- 2 Tank objects one assigned to player 1 and player 2 control schemes respectively
- Players earn 1 point by shooting opposing player tank
- Keyboard interaction
- Timefix: 60 frames pr. second

keyPressed() {

//Player 2

Player 2 object has following control scheme:

1. Press “q” to shoot one bullet. If held, shoot additional bullets at the point of time where bullet.duration is exceeded for the last shot made, or if previous bullet collides with wall while bullet.bounce = false.

2. Hold “w” to move forward and maintain moving forward with fixed speed. One quick press does not move tank, but holding it in for a quarter of a second starts movement. Releasing “w” stops movement after a quarter of a second.

3. Press “q” or “e” to rotate vehicle once counterclockwise or clockwise, respectively. If held, maintain rotation with fixed time duration between turns. If trying to use both, only “e” will work.

Must be able to do all of this at once without disrupting current keyPresses or potentially keyReleases, except for rotating in both directions at the same time which is at stated above not possible.

//Player 1

Player 1 object has following control scheme:

1. Press “SPACE” to shoot one bullet. If held, shoot additional bullets at the point of time where bullet.duration is exceeded for the last shot made, or if previous bullet collides with wall while bullet.bounce = false.

2. Hold “UP_ARROW” to move forward and maintain moving forward with fixed speed. One quick press does not move tank, but holding it in for a quarter of a second starts movement. Releasing “UP_ARROW” stops movement after a quarter of a second.

3. Press “LEFT_ARROW” or “RIGHT_ARROW” to rotate vehicle counterclockwise or clockwise, respectively. If held, maintain rotation with fixed time duration between turns. If trying to use both, only “RIGHT_ARROW” will work.

Must be able to do all of this at once without disrupting other keyPresses or potentially keyReleases, except for when rotating in both directions at the same time which is as stated above not possible.

Press “I” to buildLevel using current level value, a “reset”-function

Press “O” to increment level value and then buildLevel using that value, a “go to next level”-function

}

buildLevel() {

Every time and according to each level description: reset score to 0, position each player at each side of the map, setup wall objects, update booleans.

From level 1 through 5: bullet.bounce = false; tank.invisible = false; bullet.duration = 1.5 sec;

From level 6 through 9: bullet.bounce = true;

- Level 8 through 9: bullet.duration = 2.5 sec;

From level 10 through 14: tank.invisible = true;

- Level 10 through 11: bullet.duration = 1 sec; bullet.bounce = false;
- Level 13 through 14: bullet.duration = 2 sec; bullet.bounce = true;

}

tank.collision(){

Depending on which angle player is hitting a wall from (this incidentally also corresponds to set increments or decrements for thrustSpeedX, thrustSpeedY or both), player.pos.x and/or player.pos.y is moved towards the opposite angle by value: thrustSpeed * 2. Player tank maintains current rotationAngle but is pushed back by hitting a wall.

}

tank.rotate() {

Iterate over function operations every 1/3 of a second if key is held || iterate every time function is called if player is dying

Depending on input key, instantly rotate tank 30 degrees clockwise or counterclockwise

}

tank.dyingPhase() {

Tank is positioned at a random posX and random posY. Not on a wall or another tank.

Tank rotates clockwise automatically

Both players cannot provide input to tanks while the phase is ongoing

After 2 seconds dying = false;

}

bullet.collision() {

Check for collision with opposing player - use variable name to determine who is opposing player or implement a tag.

If opposing player is hit by enemy bullet, opposing playerX.dying = true and this score.playerX_scored = true

Check for collision with walls

If wall is hit && bullet.bounce = false, bullet expire()

If wall is hit && bullet.bounce = true, bullet will bounce off wall.

If rotationAngle is either 0, 90, 180 or 270, upon impact with wall

If bullet hits either left or right side of a wall, velX and rotationAngle is reversed. rotationAngle is additionally decreased by 30.

If bullet hits either up or down side of a wall, velY and rotationAngle is reversed. rotationAngle is additionally decreased by 30

If bullet hits either left or right side of a wall, velX and rotationAngle is reversed.

If bullet hits either up or down side of wall, velY and rotationAngle is reversed.

}

bullet.move() {

Corresponding to tank.rotationAngle add velX and velY to bullet.posX and Y so that it matches the direction of rotationAngle.

If bounce = false, rotationAngle is updated with any changes to player.rotationAngle.

If bounce = true, rotationAngle is fixed to original value when creating the object or until hitting wall.

}

A note about data types:

Given the resolution, all x and y coordinates are to be integers and this create the sensation that the tank jumps a block for every increment to movement

A note about collision detection:

I have chosen to keep collision detection as part of the individual objects in order to emphasize the class specific functionality that activates as part of either tank or bullet colliding with either wall or tank/bullet. It is also possible, as is shown in my Mega-Game, to put the actual collision detection into its own object as a collider and still keep class specific

functionality as part of the object. This may be regarded as more efficient since it reuses same code.

A note about update-functions:

I have thought of update functions as ordered sequences. They are called on every frame, but depending on current status of player, the order makes it certain the player dies before shooting or moving around. It may not make a huge difference in this small game, but it does make sure that there is a priority to updating a dying phase which determines whether you can move around or not as well. In general, collision is prioritized before actions and this is also relevant for when a tank drives into a wall and must be stopped before driving through it.