

# LOGIC PLAY GAMES

SHAPE SHIFTERS V1.0

---

## DESIGN TRADE OFFS:

Collision was a very big thing for us. In the final game the collision doesn't work exactly as planned. We just had to deal with it. For example, when you hit a fan in the game level, collision is only detected at the center of the fan and not on its spokes. Collision was also a big problem when going through walls. We had to basically create another invisible object at each hole, then hand code one of these objects for each wall. For some reason, when we put them in a for loop, our collision subsystem (Spark's Collision) would not recognize the objects correctly.

At the beginning of game level design, we had a glitch in the level that we couldn't take care of through the collision system without making major changes. When the player's object would hit a corner of the level, the collision would actually allow the player to go outside the bounds of the level. The object would then be floating in a sea of blue while watching the rest of the level pass by. To fix this, the first thing we did was write some code to automatically put the player's object back inside the game level. That worked, but it wasn't a very good fix. The person playing the game would still realize that there was a glitch in the system because of the rapid moving of their object back to the center. The next thing we did was just add more walls, so the player's object could never get to the corner. That seemed to work very well, so we didn't worry about it anymore.

We also envisioned that playing the game with the WiiMote would be easier. The game is playable with a WiiMote, but using one doesn't necessarily make the game more fun. It does on the other hand make it more challenging though.

Our first set of background music was completely scrapped for a faster paced game.

---

## EXPLANATION OF OVERALL SOFTWARE DESIGN:

### *Data Structures:*

#### Global Variables:

- Integers
  - g\_iDisplayResolutionX = 1024
  - g\_iDisplayResolutionY = 768
  - g\_iLevel = 1 //player initial level
  - g\_iScore = 0 //player initial score
  - g\_iEnergy = 100 //Player initial energy level
  - g\_iOldPlayerObjectID = 8 //Player initial object id number
  - g\_iPlayerObjectID = 8 //Player initial object id number
  - g\_iRotateCount //Rotator variable for some game objects
  - g\_iLevelWaitTimer = 0 //Timer for time between levels
  - g\_iLevelWaitTime = 5 //Time in seconds between level loads
  - Variables used to rotate walls
    - g\_iRotateDegreeX = 0
    - g\_iRotateDegreeY = 120
    - g\_iRotateDegreeZ = 240
- Floats
  - g\_fSpeed = 4.0; //Speed of Oncoming Walls
  - g\_fObjectSpeed = 1; //Speed of Object
  - g\_fSpinning = 0;
  - g\_fCounter;
  - g\_fPlayerSpeed = 3.0;
  - g\_fPlayerSpeedLimit = 6;
  - Player Object Position Information:
    - g\_fMemoCubeX;
    - g\_fMemoCubeY;
    - g\_fMemoCubeZ;
- Booleans
  - g\_bSlowMode = false;
  - g\_bTransMode = false;

### *Finite State Machine:*

The finite state machine had x states which included gameSetup, gameTitle, gameWaitForGo, gameLevel, gameReset, gameLoading, gameLevelWait, and gameDie.

#### Main FSM

- (1) gameSetup
  - (a) set resolution

- (b) set window position
- (c) load sounds
- (d) load text for font
- (e) go to title mode
- (2) gameTitle
  - (a) load all images
  - (b) display initial menu image
  - (c) play menu background sound
- (3) gameWaitForGo
  - (a) if detect "go"
    - (i) show loading screen
    - (ii) start loading game (go to loading state)
- (4) gameLevel
  - (a) Update HUD
  - (b) Set lighting
  - (c) Move and rotate fans
  - (d) Move the player's object
  - (e) Fix going outside of walls
  - (f) Movement (associates left, right, up, and down with corresponding keys on keyboard)
  - (g) Move the orbiting spheres
  - (h) Switch objects
    - (i) If player presses "a" then change to cube
    - (ii) If player presses "s" then change to pyramid
    - (iii) If player presses "d" then change to sphere
  - (i) Speed up orbiting spheres after transformation
  - (j) Make camera follow player object
  - (k) Collision
    - (i) Object hits fans
    - (ii) Object hits medipack
    - (iii) Object hits hourglass
    - (iv) Object hits mine
    - (v) Object hits walls
  - (l) Gain trickle of energy constantly
  - (m) Go to next level if player gets to end of current level
  - (n) Go to reset state if player presses "r" key
  - (o) Go to die state if player's energy reaches zero
- (5) gameReset
  - (a) delete all objects
  - (b) go to title screen
- (6) gameLoading
  - (a) Start Sparky Collision
  - (b) Setup textures
  - (c) Create oncoming walls
  - (d) Setup collision for oncoming walls and side walls

- (e) Create power ups inside the level
  - (f) Rotate the oncoming walls
  - (g) Create rotating energy spheres
  - (h) Move the camera to the initial player object
  - (i) Create the big doors
  - (j) Start/ stop sounds
  - (k) Go to game level state
- (7) gameLevelWait
- (a) timer to wait for next level
  - (b) reset player to beginning of level
  - (c) recreate all the powerups
  - (d) go to gameLevel
- (8) gameDie
- (a) show gameover screen
  - (b) Go to reset state if player presses “r” key

### *Graphics:*

Everything we used was built into the engine, except the objects. The powerups, walls, and particles around the player object were created using blender.

### *AI:*

There is no AI.

### *Sounds:*

All the sounds are handled in a separate file that’s called from main.cpp. This file loads all the sounds, and determines which background music to play depending on what level the game is in.

### *Main Game Engine:*

We chose DarkGDK because it was in C++, which is a language that is familiar to everyone in the group.

---

## EXPLANATIONS OF HOW SPECIFIC VISUAL AND AUDIO EFFECTS WERE ACHIEVED:

### SOUNDS:

- ShapeShift – the sound you hear when changing shapes
  - This sound was made using Adobe Audition to record our team lead making a swoosh sound with his mouth. An echo effect was then added to give the sound a more futuristic feel.
- Background Music
  - Odd Levels
    - The background music for odd levels was made using Garageband and the keyboard piano built into it. It also contains a built in beat.
  - Even Levels
    - The background music for even levels of the game was made using Sony's Acid 6.0. All the sounds in the second levels back ground music were built in sounds just arranged in a different order.
  - Menu/Title Screen
    - This sound is the tractor operating sound from the course website, with a sweeping effect in the background to make it have a more ominous feel to it.
- Powerups
  - Health
    - The sound you hear when getting the red spinning cross in the level was made using Garageband with some help from Dr. Beck. He basically cut a short tone, then looped it for about half a second.
  - SloMo
    - The SloMo sound was also created using GarageBand. It was originally supposed to be the death screen sound, but we found that it fit the SloMo effect so we used it there.
  - Mines
    - The explosion sound for the mines was taken from a website that offered free explosion sounds because at that point we were running out of time. Here's a link to the website: <http://www.mediacollege.com/downloads/sound-effects/explosion/>
- Wall Shock
  - The shock effect came from the sample pack off of the course website.
- Door Open Collage
  - The opening door sounds also came from one of the effects in the pack from the course website.

## VISUALS:

- Walls
  - Advancing
    - Originally the walls would move towards the player, but to give the object a feel of falling, the object now moves forward.
  - Changing hole position
    - Originally we planned on making the holes show up in totally random locations every time the game is started. But after seeing how fast paced the game has to be in order to be fun, we implemented a fake random algorithm that only spins some walls at a certain angle to have a limited number of hole locations. That way the player can expect a certain level of consistency.
- Shape Shifting
  - The feel of gaining energy while shape shifting was done by creating a bunch of spheres that would increase/decrease in size and speed as they went straight from one point to another. To create so many, we just made one then loaded twenty copies of the same object into the game which all rotate around the player object at different angles giving it an electron-feel.
- Power-ups
  - All power-ups and enemies were created and animated with Blender and imported into the game as .x objects. When importing such rotating objects into DarkGDK, the objects actually pulsate. We actually liked that 'feature/bug' for our power-ups to make them more noticeable.
- Player Objects
  - The three player shapes were created with simple geometric polygons in DarkGDK and then textures were applied.