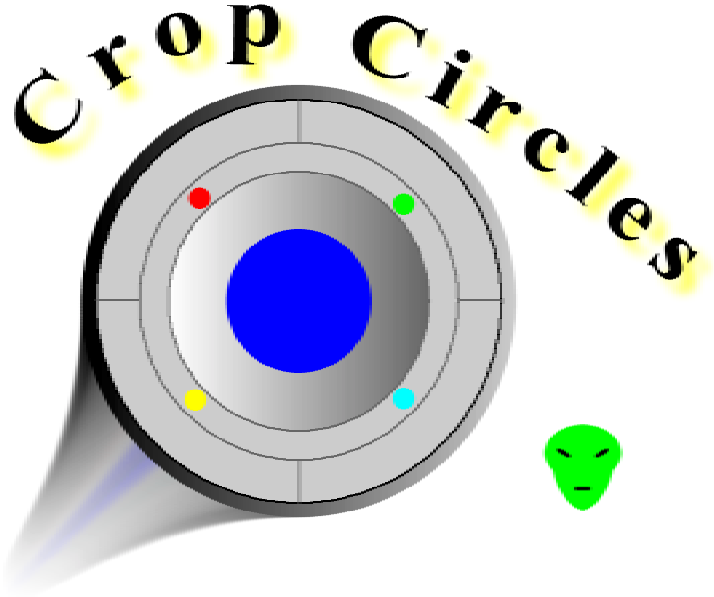
FINAL PROJECT REPORT - "CROP CIRCLES" GAME

AUTHOR: Steve Baer
DATE: December 17, 2001



UW
BOTHELL
UNIVERSITY OF
WASHINGTON

INDEX PAGE

USER MANUAL 2
Overview 2
Story Line 2
Game Play 2
User Interface 3
Game Play Area 3
World View 3
Accelerator Controls 4
Game Controls 4

GAME SYSTEM DESIGN 5
Overall Game Class Structure 5
CC_GameModel (MODEL) 5
CC_GUI (VIEW) 5
CC_Controller (CONTROLLER) 5
Model Class Structure 5
Game State Transition Method 6
Game State Constants 6
void switchGameState(GameState) 6
void callSimulator() 6
Limitations 6
Future Version Features 7
Character Enhancements 7
Playability 7
Collision Detection 7

USER MANUAL

Overview
Crop Circles is a two dimensional video game designed to work on Win32 compatible computers. The game requires windows 95 or higher and a graphic cards with openGL capability.

Story Line
It's New Years Eve on Mars. After a night of celebration, someone came up with the brilliant idea of taking some spaceships to earth and making crop circles (Martians like to make crop circles and laugh at the confused earthlings.) This joyride ended up not being such a great idea because most of the Martians were very intoxicated and crashed their ships.

It is your job to collect the stranded and hung over Martians before they are swept up in tornadoes, caught by the FBI for extensive testing, or killed by some other danger of earth.

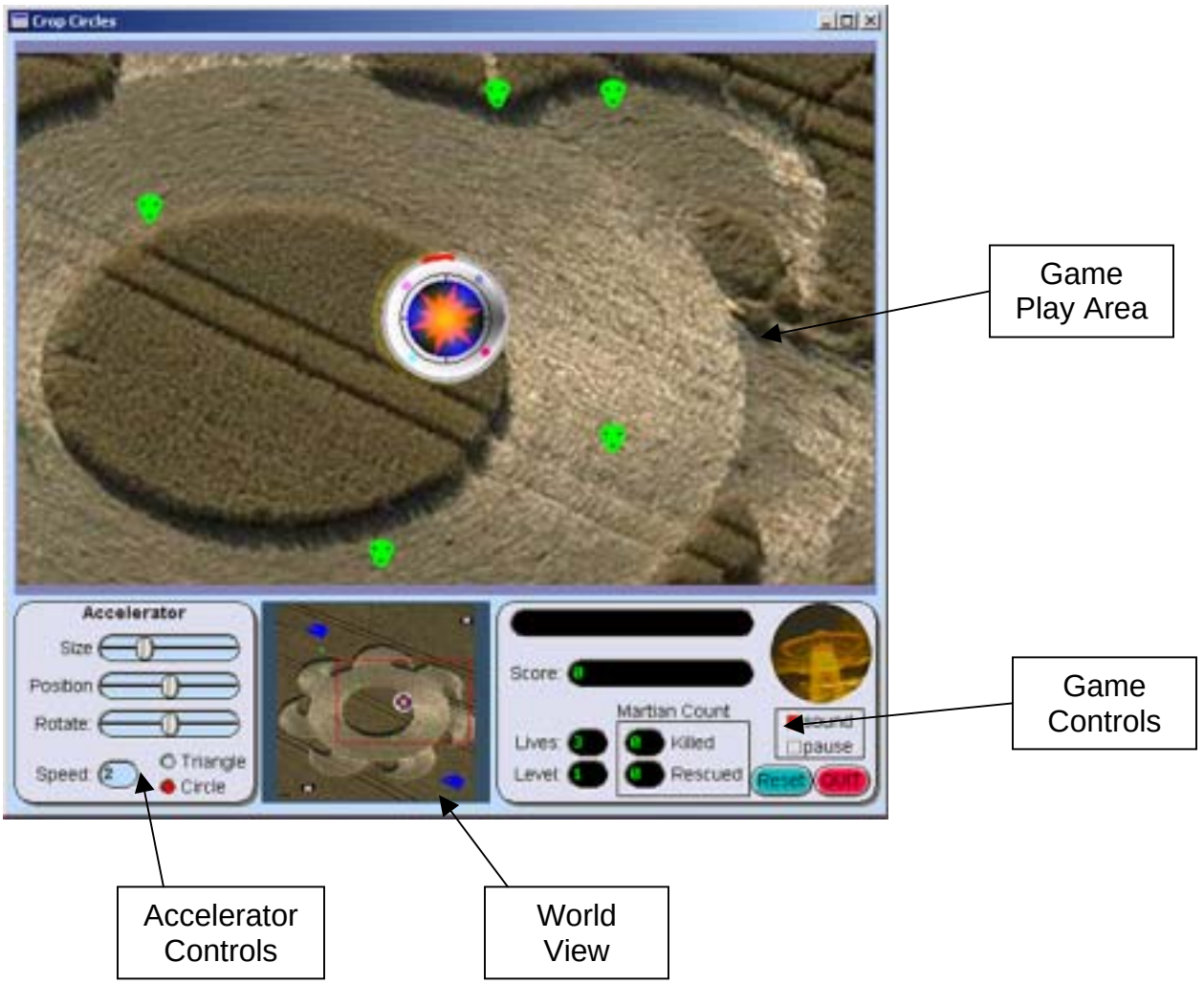
Game Play
A player pilot's his/her spaceship around a 2-D rectangular world with the goal of collecting Martians. A level is complete when all the Martians have either been collected or killed by enemy objects. Each new level contains more Martians along with faster enemies. The game's keyboard controls are as follows:

Key	Purpose
Up Arrow	Change spaceship direction to up
Down Arrow	Change spaceship direction to down
Left Arrow	Change spaceship direction to left
Right Arrow	Change spaceship direction to right
S	Decrease speed of spaceship
F	Increase speed of spaceship
Esc	Pause / Unpause game
F1	Zoom out
F2	Zoom in

- The spaceship will automatically collect a Martian when it comes in contact with a Martian.
- Avoid police cars and tornados. Contact with these guys will cause you to lose a life.
- Touching the edge of a crop circle boundary will cause the spaceship to bounce in the opposite direction.

User Interface

Crop Circle's user interface is comprised of a single window with four main components.



Game Play Area

This is a viewport that shows a portion of a level's world. The spaceship is always visible and centered inside this viewport. As the spaceship moves, the portion of the world that the game play area focuses on adjusts (pans). The player uses this area to play the game.

World View

A viewport that shows an entire level's world. A red rectangle on the worldview represents what the player sees in the game play area. The worldview is radar for the user to view Martians and enemies that are outside of the game play area. Pressing and moving the right mouse button over the world view will allow for panning around the game play area's bounds. Holding the middle mouse button down and moving left will zoom in on the world while moving right will zoom out on the world.

Accelerator Controls

These are a small set of controls for adjusting the appearance, size, and location of the space ship's accelerators. Scale, position, and rotation of the accelerators are adjusted by the slider controls. The accelerators may be represented as circles or triangles by selecting the appropriate radio button.

Game Controls

This box displays the current game information (score, lives left, current level). This set of GUI controls also contains a pause, mute, reset and quit button.

GAME SYSTEM DESIGN

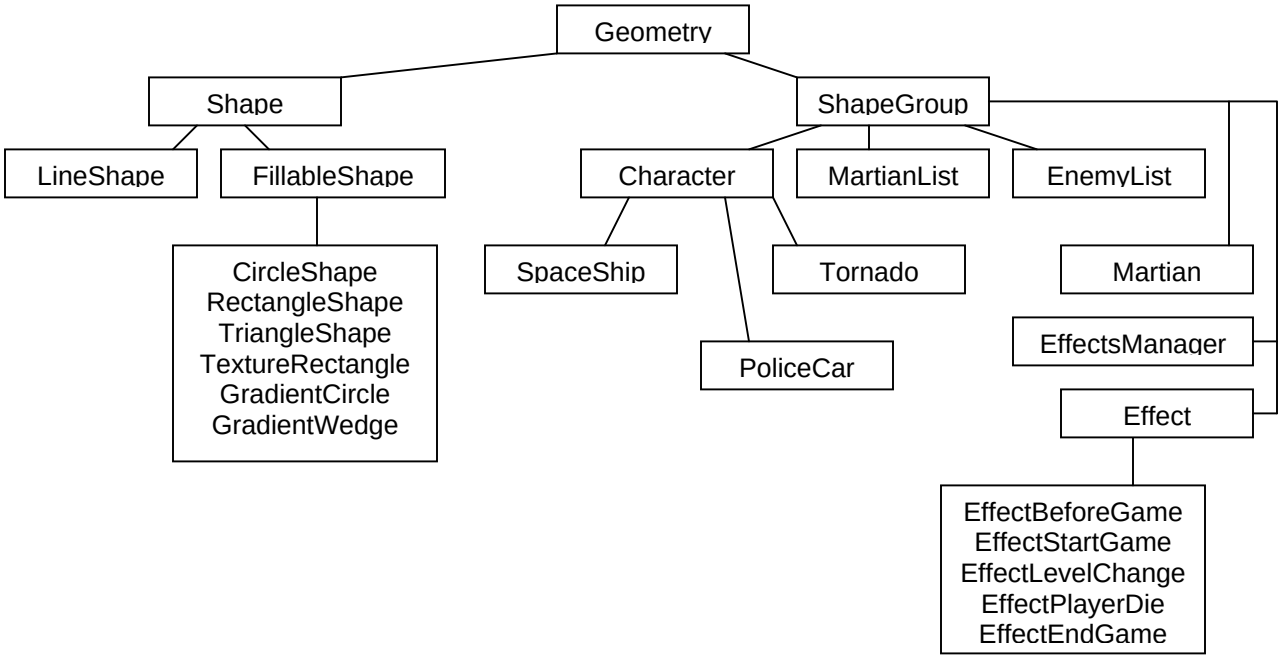
Overall Game Class Structure
Crop Circles uses a modified Model-View-Controller architecture. The classes that implement this architecture are CC_GameModel, CC_GUI, and CC_Controller.

CC_GameModel (MODEL)
The game contains a single CC_GameModel object. This object contains all of the game's shapes and states. When requested by the conroller, the game updates it's state and draws all of it's shapes to the view.

CC_GUI (VIEW)
The GUI class contains two DrawableArea objects (my implementation of an OpenGL viewport) and multiple FLTK widgets. The GUI was designed using the Fluid program.

CC_Controller (CONTROLLER)
This class is actually a group of static functions that provide a connection between the model and the view. The controller contains a GLUT timer function that notifies the model to update itself every 35 milliseconds.

Model Class Structure
The CC_GameModel class contains a shapegroup object that is redrawn every time the redraw function is called. This ShapeGroup contains a background (TextureRectangle), spaceship, group of martians (MartianList), group of enemies (EnemyList), and a visual effect (EffectsManager) object. These objects are described below:



Game State Transition Method

The CC_GameModel object's callSimulator() funtion is called every 35 milliseconds by the controller. The model changes itself based on the current game state. Game state is handled by an enumeration of game state constants and two functions.

Game State Constants

CC_GameModel contains an enumeration of states. The game is always in one (and only one) of these states. The states are:

1. Before Game – The game has not started and the player has not yet told the game that he/she wants to begin playing.
2. Start Game – The player has told the game that he/she wishes to begin playing.
3. Plating – Normal game playing is occurring.
4. Game Over – The player has lost all of his/her lives.
5. Level Change – All of the Martians are gone on the current level. The player is moving to a new level.
6. Player Die – The spaceship has collided with an enemy object.

void switchGameState(GameState)

This function is called when the game is changing from its current game state to a new game state. This allows for loading of new visual effects, sounds to be played, and notification of model objects to reinitialize themselves.

void callSimulator()

The call simulator function is the heart of the CC_GameModel class. This function is one large switch statement that performs that appropriate task based on the game's current state. With many states, this is simply a call to the current visual effect to update itself. If the effect is finished updating, then change to the next appropriate game state. In the playing state, this function tells all the objects to update their positions and check for collisions.

Limitations

The Crop Circles gaming system's primary limitation is maximum character count. Collision detection is performed with a $O(n^2)$ algorithm. If the character count were increased to a high number, the game would probably start missing its "frame rate".

Future Version Features

The current version of Crop Circles has a strong overall game structure. The areas that need the most work are character enhancements, playability, and improved collision detection.

Character Enhancements

- The spaceship (hero) character has a range of five speeds and four directions of movement. Future versions should allow for travel in any direction (possibly through use of a joystick). The ship should also have an array of weapons and Martian capturing devices.
- Enemy object movement is dim-witted at best. Some level of artificial intelligence should be built into the enemy's movement. It's understandable that a tornado travels along a random path, but police cars should at least follow some sort of driving path.
- Multiple new enemies with each new level. Possibly enemy ships or angry farmers.
- Martians should also have some sort of intelligence and movement. Maybe the Martians could slowly chase the ship around the world or attempt to fight off enemies.

Playability

- Sound is an integral part of a good video game. A background soundtrack and different sound effects would enrich the game.
- More visual effects for events occurring in the game.
- Provide ability to work with joystick.
- A better camera algorithm for the game play area. Having the game play area pan in order to keep the ship centered can provide a somewhat sickening effect for the player. A better method would be to pan the play area when the ship begins to get close to one edge of the play area.
- The world view should be modified so it acts more as a radar. It should only show the spaceship and martians (not enemies.) This increases the difficulty of the game and makes the players look at the play area instead of "cheating" by just watching the world view all the time.
- Add a save game and high score hall of fame.

Collision Detection

- Characters in the game use a X-Y axis aligned bounding box for collision detection. A tighter "character" axis aligned bounding box would provide for much more realistic collisions.
- Each character should have a mass and possible spring constant. This would allow for very realistic looking bounces when collisions occur.
- Some sort of assisting structure should be developed to help speed up collision detection. Current implementation performs a check all objects for collision routine, which is very inefficient (n^2).