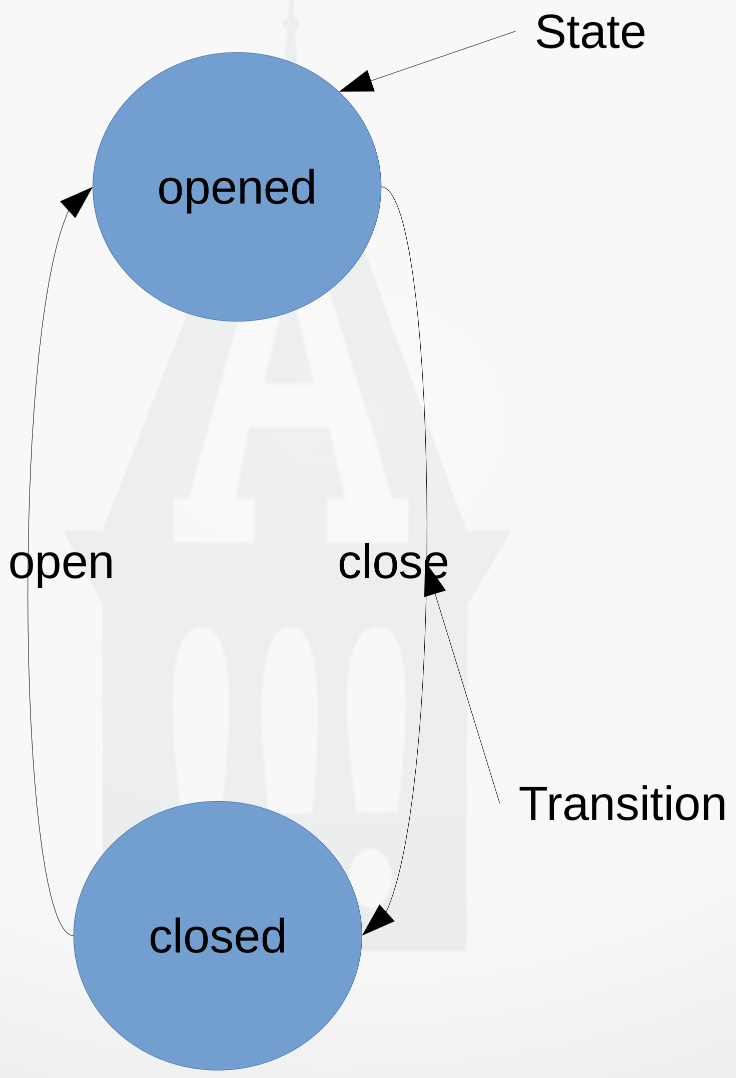
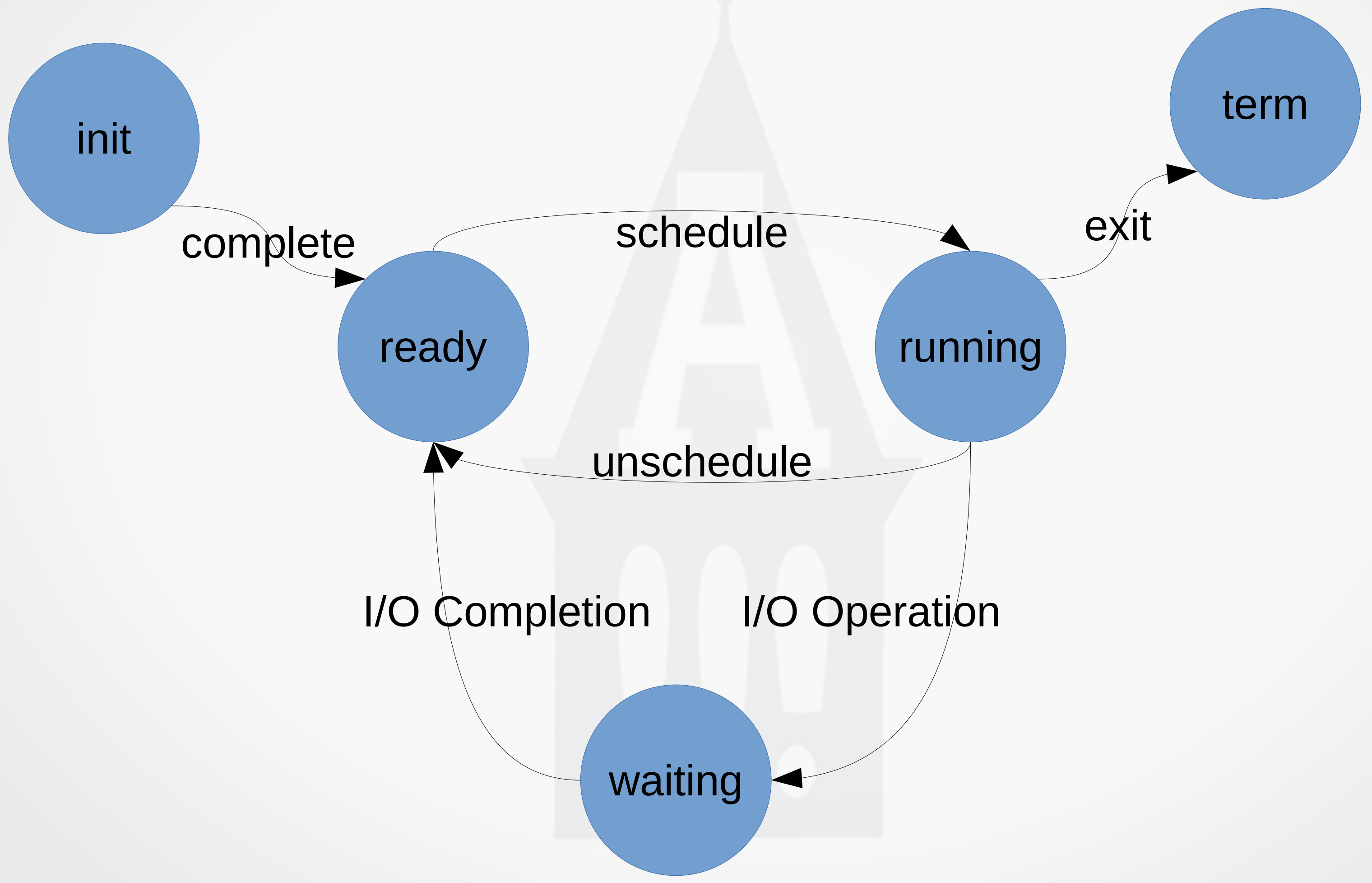
Game State Management

Internal Game & Menu State Management

Finite State Machines



Another Example



State Transition Table

State/Input	init	ready	running	waiting	term
complete	ready	...	...	...	...
schedule	...	running	...	...	...
unschedule	...	...	ready	...	...
I/O Call	...	...	waiting	...	...
I/O Finish	...	...	...	ready	...
exit	...	...	term	...	...

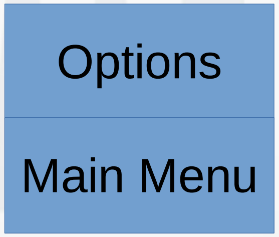
Menu State - Stacks

- New Game
- Options
  - Video
    - Basic
    - Advanced
  - Audio
  - Controls
    - Keyboard
    - Mouse
    - Controller
- High Scores
- Credits
- Exit

Main Menu

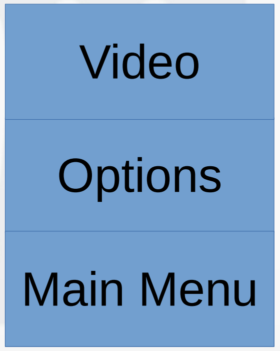
Menu State - Stacks

- New Game
- Options
  - Video
    - Basic
    - Advanced
  - Audio
  - Controls
    - Keyboard
    - Mouse
    - Controller
- High Scores
- Credits
- Exit



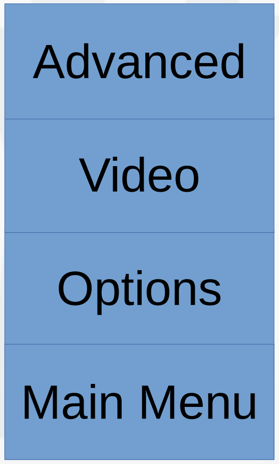
Menu State - Stacks

- New Game
- Options
  - Video
    - Basic
    - Advanced
  - Audio
  - Controls
    - Keyboard
    - Mouse
    - Controller
- High Scores
- Credits
- Exit



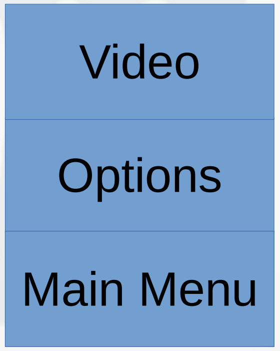
Menu State - Stacks

- New Game
- Options
  - Video
    - Basic
    - Advanced
  - Audio
  - Controls
    - Keyboard
    - Mouse
    - Controller
- High Scores
- Credits
- Exit



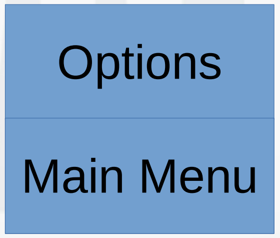
Menu State - Stacks

- New Game
- Options
  - Video
    - Basic
    - Advanced
  - Audio
  - Controls
    - Keyboard
    - Mouse
    - Controller
- High Scores
- Credits
- Exit



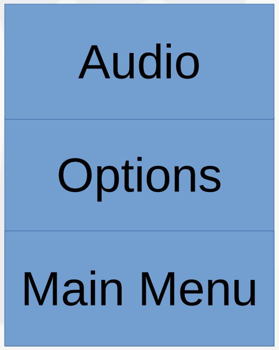
Menu State - Stacks

- New Game
- Options
  - Video
    - Basic
    - Advanced
  - Audio
  - Controls
    - Keyboard
    - Mouse
    - Controller
- High Scores
- Credits
- Exit



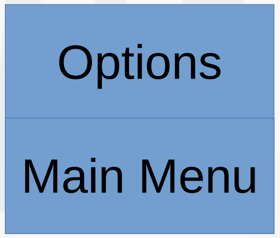
Menu State - Stacks

- New Game
- Options
  - Video
    - Basic
    - Advanced
  - Audio
  - Controls
    - Keyboard
    - Mouse
    - Controller
- High Scores
- Credits
- Exit



Menu State - Stacks

- New Game
- Options
  - Video
    - Basic
    - Advanced
  - Audio
  - Controls
    - Keyboard
    - Mouse
    - Controller
- High Scores
- Credits
- Exit



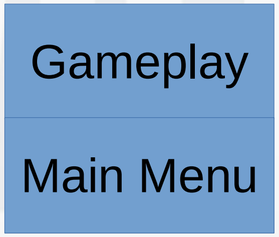
Menu State - Stacks

- New Game
- Options
  - Video
    - Basic
    - Advanced
  - Audio
  - Controls
    - Keyboard
    - Mouse
    - Controller
- High Scores
- Credits
- Exit

Main Menu

Menu State - Stacks

- New Game
- Options
  - Video
    - Basic
    - Advanced
  - Audio
  - Controls
    - Keyboard
    - Mouse
    - Controller
- High Scores
- Credits
- Exit

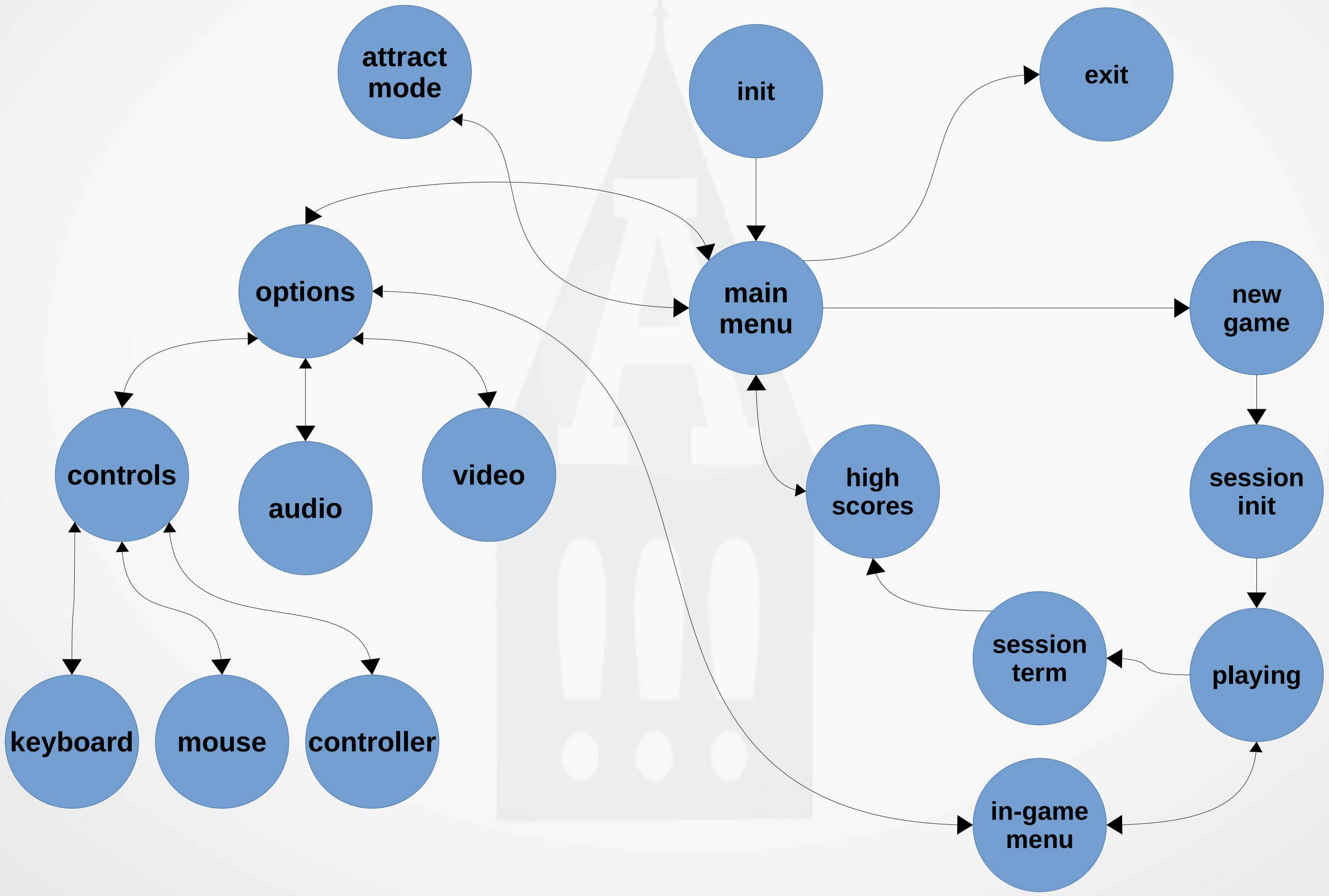


Menu State - Stacks

- New Game
- Options
  - Video
    - Basic
    - Advanced
  - Audio
  - Controls
    - Keyboard
    - Mouse
    - Controller
- High Scores
- Credits
- Exit

Main Menu

Menus – State Transition Diagram



Game State

Simple Menuing - MonoGame

Game State – IGameState

```
public interface IGameState
{
     void initialize(GraphicsDevice graphicsDevice, GraphicsDeviceManager graphics);
     void loadContent(ContentManager contentManager);
     GameStateEnum processInput(GameTime gameTime);
     void update(GameTime gameTime);
     void render(GameTime gameTime);
}
```

Note these are matches with the core “program” game loop

Game State – GameStateView

```
public interface IGameState
{
     void initialize(GraphicsDevice graphicsDevice, GraphicsDeviceManager graphics);
     void loadContent(ContentManager contentManager);
     GameStateEnum processInput(GameTime gameTime);
     void update(GameTime gameTime);
     void render(GameTime gameTime);
}
```

```
public abstract class GameStateView : IGameState
{
     protected GraphicsDeviceManager m_graphics;
     protected SpriteBatch m_spriteBatch;

     public void initialize(GraphicsDevice graphicsDevice, GraphicsDeviceManager graphics)
     {
          m_graphics = graphics;
          m_spriteBatch = new SpriteBatch(graphicsDevice);
     }
     public abstract void loadContent(ContentManager contentManager);
     public abstract GameStateEnum processInput(GameTime gameTime);
     public abstract void render(GameTime gameTime);
     public abstract void update(GameTime gameTime);
}
```

Game State – GameStateEnum

```
public interface IGameState
{
    void initialize(GraphicsDevice graphicsDevice, GraphicsDeviceManager graphics);
    void loadContent(ContentManager contentManager);
    GameStateEnum processInput(GameTime gameTime);
    void update(GameTime gameTime);
    void render(GameTime gam
}
```

```
public enum GameStateEnum
{
    MainMenu,
    GamePlay,
    HighScores,
    Help,
    About,
    Exit
}
```

```
public abstract class Gam
{
    protected GraphicsDeviceManager m_graphics;
    protected SpriteBatch m_spriteBatch;

    public void initialize(GraphicsDevice graphicsDevice, GraphicsDeviceManager graphics)
    {
        m_graphics = graphics;
        m_spriteBatch = new SpriteBatch(graphicsDevice);
    }
    public abstract void loadContent(ContentManager contentManager);
    public abstract GameStateEnum processInput(GameTime gameTime);
    public abstract void render(GameTime gameTime);
    public abstract void update(GameTime gameTime);
}
```

GameStateView - Example

```
public class AboutView : GameStateView
{
    private SpriteFont m_font;
    private const string MESSAGE = "*I* wrote this amazing game!";

    ...
}
```

GameStateView - Example

```
public class AboutView : GameStateView
{
    private SpriteFont m_font;
    private const string MESSAGE = "*I* wrote this amazing game!";


}
```

```
public override void loadContent(ContentManager contentManager)
{
    m_font = contentManager.Load<SpriteFont>("Fonts/menu");
}
```

GameStateView - Example

```
public class AboutView : GameStateView
{
    private SpriteFont m_font;
    private const string MESSAGE = "*I* wrote this amazing game!";

    ...
}
```

```
public override void loadContent(ContentManager contentManager)
{
    m_font = contentManager.Load<SpriteFont>("Fonts/menu");
}
```

```
public override GameStateEnum processInput(GameTime gameTime)
{
    if (Keyboard.GetState().IsKeyDown(Keys.Escape))
    {
        return GameStateEnum.MainMenu;
    }

    return GameStateEnum.About;
}
```

GameStateView - Example

```
public class AboutView : GameStateView
{
    private SpriteFont m_font;
    private const string MESSAGE = "*I* wrote this amazing game!";

    ...
}
```

```
public override void loadContent(ContentManager contentManager)
{
    m_font = contentManager.Load<SpriteFont>("Fonts/menu");
}
```

```
public override GameStateEnum processInput(GameTime gameTime)
{
    if (Keyboard.GetState().IsKeyDown(Keys.Escape))
    {
        return GameStateEnum.MainMenu;
    }

    return GameStateEnum.About;
}
```

```
public override void render(GameTime gameTime)
{
    m_spriteBatch.Begin();
    Vector2 stringSize = m_font.MeasureString(MESSAGE);
    m_spriteBatch.DrawString(m_font, MESSAGE,
      new Vector2(m_graphics.PreferredBackBufferWidth / 2 - stringSize.X / 2,
      m_graphics.PreferredBackBufferHeight / 2 - stringSize.Y), Color.Yellow);
    m_spriteBatch.End();
}
```

GameStateView - Example

```
public class AboutView : GameStateView
{
    private SpriteFont m_font;
    private const string MESSAGE = "*I* wrote this amazing game!";


}
```

```
public override void loadContent(ContentManager contentManager)
{
    m_font = contentManager.Load<SpriteFont>("Fonts/menu");
}
```

```
public override GameStateEnum processInput(GameTime gameTime)
{
    if (Keyboard.GetState().IsKeyDown(Keys.Escape))
    {
        return GameStateEnum.MainMenu;
    }

    return GameStateEnum.About;
}
```

```
public override void render(GameTime gameTime)
{
    m_spriteBatch.Begin();
    Vector2 stringSize = m_font.MeasureString(MESSAGE);
    m_spriteBatch.DrawString(m_font, MESSAGE,
      new Vector2(m_graphics.PreferredBackBufferWidth / 2 - stringSize.X / 2,
      m_graphics.PreferredBackBufferHeight / 2 - stringSize.Y), Color.Yellow);
    m_spriteBatch.End();
}
```

```
public override void update(GameTime gameTime)
{
}
```

Management of Game State

```
public class GameStateDemo : Game
{
    private GraphicsDeviceManager m_graphics;
    private IGameState m_currentState;
    private GameStateEnum m_nextStateEnum = GameStateEnum.MainMenu;
    private Dictionary<GameStateEnum, IGameState> m_states;

    . . .
}
```

Management of Game State

```
public class GameStateDemo : Game
{
     private GraphicsDeviceManager m_graphics;
     private IGameState m_currentState;
     private GameStateEnum m_nextStateEnum = GameStateEnum.MainMenu;
     private Dictionary<GameStateEnum, IGameState> m_states;

     . . .
}
```

Tracking of the current game state (a view)

Management of Game State

```
public class GameStateDemo : Game
{
    private GraphicsDeviceManager m_graphics;
    private IGameState m_currentState;
    private GameStateEnum m_nextStateEnum = GameStateEnum.MainMenu;
    private Dictionary<GameStateEnum, IGameState> m_states;

    . . .
}
```

Tracking of the current game state (an enum)

Management of Game State

```
public class GameStateDemo : Game
{
    private GraphicsDeviceManager m_graphics;
    private IGameState m_currentState;
    private GameStateEnum m_nextStateEnum = GameStateEnum.MainMenu;
    private Dictionary<GameStateEnum, IGameState> m_states;

    . . .
}
```

Container for the the different possible game states (enums to views)

Management of Game State

```
public class GameStateDemo : Game
{
    private GraphicsDeviceManager m_graphics;

}
```

```
protected override void Initialize()
{
    . . .

    // Create all the game states here
    m_states = new Dictionary<GameStateEnum, IGameState>();
    m_states.Add(GameStateEnum.MainMenu, new MainMenuView());
    m_states.Add(GameStateEnum.GamePlay, new GamePlayView());
    m_states.Add(GameStateEnum.HighScores, new HighScoresView());
    m_states.Add(GameStateEnum.Help, new HelpView());
    m_states.Add(GameStateEnum.About, new AboutView());

    // We are starting with the main menu
    m_currentState = m_states[GameStateEnum.MainMenu];

    . . .
}
```

Management of Game State

```
public class GameStateDemo : Game
{
    private GraphicsDeviceManager m_graphics;

}
```

```
protected override void Initialize()
{
     . . .

     // Create all the game states here
     m_states = new Dictionary<GameStateEnum, IGameState>();
     m_states.Add(GameStateEnum.MainMenu, new MainMenuView());
     m_states.Add(GameStateEnum.GamePlay, new GamePlayView());
     m
     m
     m

     /
     m

     .
}
```

```
protected override void Update(GameTime gameTime)
{
     m_nextStateEnum = m_currentState.processInput(gameTime);
     // Special case for exiting the game
     if (m_nextStateEnum == GameStateEnum.Exit)
     {
          Exit();
     }

     m_currentState.update(gameTime);

     base.Update(gameTime);
}
```

Management of Game State

```
public class GameStateDemo : Game
{
     private GraphicsDeviceManager m_graphics;



}
```

```
protected override void Initialize()
{
     . . .

     // Create all the game states here
     m_states = new Dictionary<GameStateEnum, IGameState>();
     m_states.Add(GameStateEnum.MainMenu, new MainMenuView());
     m_states.Add(GameStateEnum.GamePlay, new GamePlayView());
     m
     m
     m

     /
     m

     .
}
```

```
protected override void Update(GameTime gameTime)
{
     m_nextStateEnum = m_currentState.processInput(gameTime);
     // Special case for exiting the game
     if (m_nextStateEnum == GameStateEnum.Exit)
     {
          Exit();
     }

     m_cu

     base
}
```

```
protected override void Draw(GameTime gameTime)
{
     GraphicsDevice.Clear(Color.Black);

     m_currentState.render(gameTime);

     m_currentState = m_states[m_nextStateEnum];

     base.Draw(gameTime);
}
```

Game State

Inspection of Demo