```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;
 5  using System.Threading.Tasks;
 6  using DragonBallGame.CharacterClass;
 7  using DragonBallGame.TransformationState;
 8
 9  namespace DragonBallGame.CharacterClass
10  {
11      public abstract class Character
12      {
13          private string _name;
14          private int _power;
15          private int _health;
16          private int _maxHealth;
17          private string _specialAbility;
18          private int _transformationLvl;
19          private int _maxLevel;
20          private string _form;
21          private int _energy;
22          private int _maxEnergy = 100;
23          private ITransformationState _transformationState;
24          private bool _isBlocking;
25
26          public Character(string name, int power, int health, string
              specialAbility)
27          {
28              Name = name;
29              Power = power;
30              Health = health;
31              _maxHealth = health;
32              SpecialAbility = specialAbility;
33              TransformationLevel = 0;
34              _form = "Base Form";
35              _energy = 0;
36              _transformationState = new SuperSaiyan1();
37              _isBlocking = false; // Initialize as not blocking
38          }
39
40          public string Name { get => _name; set => _name = value; }
41          public int Power { get => _power; set => _power = value; }
42          public int Health { get => _health; set => _health = value; }
43          public int MaxHealth { get => _maxHealth; set => _maxHealth =
              value; }
44          public string SpecialAbility { get => _specialAbility; set =>
              _specialAbility = value; }
45          public int TransformationLevel { get => _transformationLvl; set =>
               _transformationLvl = value; }
```

```
46          public abstract int MaxLevel { get; }
47          public string Form { get => _form; set => _form = value; }
48
49          public int Energy
50          {
51              get => _energy;
52              set
53              {
54                  if (value > _maxEnergy)
55                      _energy = _maxEnergy;
56                  else if (value < 0)
57                      _energy = 0;
58                  else
59                      _energy = value;
60              }
61          }
62
63          public bool IsBlocking
64          {
65              get { return _isBlocking; }
66          }
67
68          // Method to start blocking
69          public void Block()
70          {
71              _isBlocking = true;
72          }
73
74          // Method to stop blocking
75          public void StopBlocking()
76          {
77              _isBlocking = false;
78          }
79
80          public void IncreaseEnergy(int amount) => Energy += amount;
81
82          public void ResetEnergy() => Energy = 0;
83
84          public virtual void OnDuplicateRecruited()
85          {
86              _transformationState.Handle(this);
87          }
88
89          public void SetTransformationState(ITransformationState
              transformationState)
90          {
91              _transformationState = transformationState;
92          }
93
```

```
 94          public void ResetHealth()
 95          {
 96              _health = _maxHealth;
 97              StopBlocking(); // Ensure blocking is reset when health is
                   reset
 98          }
 99      }
100  }
101
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;
using DragonBallGame.TransformationState;

namespace DragonBallGame.CharacterClass
{
    public class Goku : Character
    {
        public Goku() : base("Son Goku", 950, 500, "Kamehameha")
        {
        }

        public override int MaxLevel => 5;

        public override void OnDuplicateRecruited()
        {
            base.OnDuplicateRecruited();
            switch (TransformationLevel)
            {
                case 0:
                    SetTransformationState(new SuperSaiyan1());
                    break;
                case 1:
                    SetTransformationState(new SuperSaiyan2());
                    break;
                case 2:
                    SetTransformationState(new SuperSaiyan3());
                    break;
                case 3:
                    SetTransformationState(new SuperSaiyanGod());
                    break;
                case 4:
                    SetTransformationState(new SuperSaiyanBlue());
                    break;
                case 5:
                    SetTransformationState(new UltraInstinct());
                    break;
                default:
                    break;
            }
        }
    }
}
```

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using DragonBallGame.TransformationState;

namespace DragonBallGame.CharacterClass
{
    public class Vegeta : Character
    {
        public Vegeta() : base("Vegeta", 1000, 500, "Galick Gun")
        {
        }

        public override int MaxLevel => 4;

        public override void OnDuplicateRecruited()
        {
            base.OnDuplicateRecruited();

            switch (TransformationLevel)
            {
                case 0:
                    SetTransformationState(new SuperSaiyan1());
                    break;
                case 1:
                    SetTransformationState(new SuperSaiyan2());
                    break;
                case 2:
                    SetTransformationState(new SuperSaiyanGod());
                    break;
                case 3:
                    SetTransformationState(new SuperSaiyanBlue());
                    break;
                case 4:
                    SetTransformationState(new UltraEgo());
                    break;
                default:
                    break;
            }
        }
    }
}

```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using DragonBallGame.TransformationState;
7
8  namespace DragonBallGame.CharacterClass
9  {
10     public class Gohan : Character
11     {
12         public Gohan() : base("Son Gohan", 1050, 550, "Masenko")
13         {
14         }
15
16         public override int MaxLevel => 3;
17
18         public override void OnDuplicateRecruited()
19         {
20             base.OnDuplicateRecruited();
21
22             switch (TransformationLevel)
23             {
24                 case 0:
25                     SetTransformationState(new SuperSaiyan1());
26                     break;
27                 case 1:
28                     SetTransformationState(new SuperSaiyan2());
29                     break;
30                 case 2:
31                     SetTransformationState(new Ultimate());
32                     break;
33                 case 3:
34                     SetTransformationState(new Beast());
35                     break;
36                 default:
37                     break;
38             }
39         }
40     }
41 }
42
```

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DragonBallGame.CharacterClass
{
    public class Frieza : Character
    {
        public Frieza() : base("Frieza", 1000, 500, "Death Beam") { }

        public override int MaxLevel => 0;
    }
}

```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace DragonBallGame.CharacterClass
8  {
9      public class Cell : Character
10     {
11         public Cell() : base("Cell", 1250, 800, "Kamehameha") { }
12
13         public override int MaxLevel => 0;
14     }
15 }
16
```

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DragonBallGame.CharacterClass
{
    public class Buu : Character
    {
        public Buu() : base("Buu", 1800, 1000, "Vanishing Ball") { }

        public override int MaxLevel => 0;
    }
}

```

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DragonBallGame.CharacterClass
{
    public class Black : Character
    {
        public Black() : base("Black", 2500, 1500, "Fierce God Slicer") { }

        public override int MaxLevel => 0;
    }
}

```

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using DragonBallGame.CharacterClass;

namespace DragonBallGame.TransformationState
{
    public interface ITransformationState
    {
        void Handle(Character character);
    }
}

```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using DragonBallGame.CharacterClass;
7
8  namespace DragonBallGame.TransformationState
9  {
10     public class SuperSaiyan1 : ITransformationState
11     {
12         public void Handle(Character character)
13         {
14             character.Power += 100;
15             character.MaxHealth += 100;
16             character.Health = character.MaxHealth;
17             character.TransformationLevel++;
18             character.Form = "Super Saiyan 1";
19         }
20     }
21 }
22
```

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using DragonBallGame.CharacterClass;

namespace DragonBallGame.TransformationState
{
    public class SuperSaiyan2 : ITransformationState
    {
        public void Handle(Character character)
        {
            character.Power += 200;
            character.MaxHealth += 150;
            character.Health = character.MaxHealth;
            character.TransformationLevel++;
            character.Form = "Super Saiyan 2";
        }
    }
}

```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using DragonBallGame.CharacterClass;
7
8  namespace DragonBallGame.TransformationState
9  {
10     public class SuperSaiyan3 : ITransformationState
11     {
12         public void Handle(Character character)
13         {
14             character.Power += 300;
15             character.MaxHealth += 200;
16             character.Health = character.MaxHealth;
17             character.TransformationLevel++;
18             character.Form = "Super Saiyan 3";
19         }
20     }
21 }
22
23
```

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using DragonBallGame.CharacterClass;

namespace DragonBallGame.TransformationState
{
    public class Ultimate : ITransformationState
    {
        public void Handle(Character character)
        {
            character.Power += 400;
            character.MaxHealth += 300;
            character.Health = character.MaxHealth;
            character.TransformationLevel++;
            character.Form = "Ultimate";
        }
    }
}

```

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using DragonBallGame.CharacterClass;

namespace DragonBallGame.TransformationState
{
    public class SuperSaiyanGod : ITransformationState
    {
        public void Handle(Character character)
        {
            character.Power += 400;
            character.MaxHealth += 250;
            character.Health = character.MaxHealth;
            character.TransformationLevel++;
            character.Form = "Super Saiyan God";
        }
    }
}


```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using DragonBallGame.CharacterClass;
7
8  namespace DragonBallGame.TransformationState
9  {
10     public class SuperSaiyanBlue : ITransformationState
11     {
12         public void Handle(Character character)
13         {
14             character.Power += 500;
15             character.MaxHealth += 300;
16             character.Health = character.MaxHealth;
17             character.TransformationLevel++;
18             character.Form = "Super Saiyan Blue";
19         }
20     }
21 }
22
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using DragonBallGame.CharacterClass;
7
8  namespace DragonBallGame.TransformationState
9  {
10     public class UltraInstinct : ITransformationState
11     {
12         public void Handle(Character character)
13         {
14             character.Power += 1000;
15             character.MaxHealth += 350;
16             character.Health = character.MaxHealth;
17             character.TransformationLevel++;
18             character.Form = "Ultra Instinct";
19         }
20     }
21 }
22
```

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using DragonBallGame.CharacterClass;

namespace DragonBallGame.TransformationState
{
    public class UltraEgo : ITransformationState
    {
        public void Handle(Character character)
        {
            character.Power += 1000;
            character.MaxHealth += 400;
            character.Health = character.MaxHealth;
            character.TransformationLevel++;
            character.Form = "Ultra Ego";
        }
    }
}


```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using DragonBallGame.CharacterClass;
7
8  namespace DragonBallGame.TransformationState
9  {
10     public class Beast : ITransformationState
11     {
12         public void Handle(Character character)
13         {
14             character.Power += 1000;
15             character.MaxHealth += 500;
16             character.Health = character.MaxHealth;
17             character.TransformationLevel++;
18             character.Form = "Beast";
19         }
20     }
21 }
22
```

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using DragonBallGame.CharacterClass;

namespace DragonBallGame
{
    public class Player
    {
        private int _zeni;
        private List<Character> _recruitedCharacters;

        public Player()
        {
            _zeni = 100; // Initial Zeni
            _recruitedCharacters = new List<Character>();
        }

        public int Zeni
        {
            get { return _zeni; }
        }

        public void AddZeni(int amount)
        {
            _zeni += amount;
        }

        public void DeductZeni(int amount)
        {
            _zeni -= amount;
        }

        public void AddRecruitedCharacter(Character character)
        {
            _recruitedCharacters.Add(character);
        }

        public Character GetRecruitedCharacter(string name)
        {
            return _recruitedCharacters.Find(c => c.Name == name);
        }

        public List<Character> RecruitedCharacters
        {
            get { return _recruitedCharacters; }
        }
```

```
50      }
51
52  }
53
```

```
1  using DragonBallGame.CharacterClass;
2  using System;
3
4
5  namespace DragonBallGame
6  {
7      public static class CharacterFactory
8      {
9          public static Character CreateCharacter(string characterType)
10         {
11             switch (characterType)
12             {
13                 case "Goku":
14                     return new Goku();
15                 case "Vegeta":
16                     return new Vegeta();
17                 case "Gohan":
18                     return new Gohan();
19                 case "Frieza":
20                     return new Frieza();
21                 case "Cell":
22                     return new Cell();
23                 case "Buu":
24                     return new Buu();
25                 case "Black":
26                     return new Black();
27                 default:
28                     throw new ArgumentException("Invalid character type");
29             }
30         }
31     }
32 }
33
```

```
1  using DragonBallGame.CharacterClass;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace DragonBallGame
9  {
10
11     public class RecruitResult
12     {
13         public RecruitStatus Status { get; set; }
14         public Character Character { get; set; }
15     }
16
17     public enum RecruitStatus
18     {
19         NotEnoughZeni,
20         NoAvailableCharacters,
21         NewCharacterRecruited,
22         CharacterEvolved
23     }
24 }
25
```

```
 1  using System;
 2  using System.Collections.Generic;
 3  using DragonBallGame.CharacterClass;
 4
 5  namespace DragonBallGame
 6  {
 7      public class RecruitSystem
 8      {
 9          private static RecruitSystem _instance;
10          private List<Character> _recruitedCharacters;
11          private Random _random;
12
13          private RecruitSystem()
14          {
15              _recruitedCharacters = new List<Character>();
16              _random = new Random();
17          }
18
19          public static RecruitSystem Instance
20          {
21              get
22              {
23                  if (_instance == null)
24                  {
25                      _instance = new RecruitSystem();
26                  }
27                  return _instance;
28              }
29          }
30
31          public RecruitResult RecruitRandomCharacter(Player player)
32          {
33              if (player.Zeni < 100)
34              {
35                  return new RecruitResult
36                  {
37                      Status = RecruitStatus.NotEnoughZeni
38                  };
39              }
40
41              List<Character> availableCharacters = GetAvailableCharacters
                  (player);
42
43              if (availableCharacters.Count == 0)
44              {
45                  return new RecruitResult
46                  {
47                      Status = RecruitStatus.NoAvailableCharacters
48                  };
```

```
49              }
50
51              player.DeductZeni(100);
52
53              // Select a random character
54              int randomIndex = _random.Next(availableCharacters.Count);
55              Character randomCharacter = availableCharacters[randomIndex];
56              Character existingCharacter = player.GetRecruitedCharacter
                  (randomCharacter.Name);
57
58              if (existingCharacter != null)
59              {
60                  existingCharacter.OnDuplicateRecruited();
61                  return new RecruitResult
62                  {
63                      Status = RecruitStatus.CharacterEvolved,
64                      Character = existingCharacter
65                  };
66              }
67              else
68              {
69                  return new RecruitResult
70                  {
71                      Status = RecruitStatus.NewCharacterRecruited,
72                      Character = randomCharacter
73                  };
74              }
75          }
76
77
78          private List<Character> GetAvailableCharacters(Player player)
79          {
80              List<Character> allCharacters = new List<Character>
81              {
82                  new Goku(),
83                  new Vegeta(),
84                  new Gohan()
85              };
86
87              List<Character> availableCharacters = new List<Character>();
88
89              foreach (Character character in allCharacters)
90              {
91                  Character recruitedCharacter =
                      player.RecruitedCharacters.Find(c => c.Name ==
                      character.Name);
92
93                  if (recruitedCharacter == null ||
                      recruitedCharacter.TransformationLevel <=
```

```
                    recruitedCharacter.MaxLevel)
 94                 {
 95                     availableCharacters.Add(character);
 96                 }
 97             }
 98
 99             return availableCharacters;
100         }
101
102     }
103 }
```

```
1  using SplashKitSDK;
2  using DragonBallGame.CharacterClass;
3
4  namespace DragonBallGame
5  {
6      public class BattleManager
7      {
8          private Player _player;
9          private Window _window;
10
11
12         public BattleManager(Player player, Window window)
13         {
14             _player = player;
15             _window = window;
16         }
17
18         public string GetSelectedDifficulty()
19         {
20             while (true)
21             {
22                 SplashKit.ProcessEvents();
23                 SplashKit.ClearScreen(Color.White);
24
25                 // Draw Difficulty Selection UI
26                 SplashKit.DrawText("Select Difficulty:", Color.Black, 300,
                     150);
27
28                 // Draw Difficulty Buttons
29                 Rectangle easyButton = SplashKit.RectangleFrom(300, 200,
                     200, 40);
30                 Rectangle mediumButton = SplashKit.RectangleFrom(300, 260,
                     200, 40);
31                 Rectangle hardButton = SplashKit.RectangleFrom(300, 320,
                     200, 40);
32                 Rectangle extremeButton = SplashKit.RectangleFrom(300, 380,
                      200, 40);
33
34                 SplashKit.FillRectangle(Color.LightGreen, easyButton);
35                 SplashKit.DrawText("Easy", Color.Black, 385, 215);
36
37                 SplashKit.FillRectangle(Color.Yellow, mediumButton);
38                 SplashKit.DrawText("Medium", Color.Black, 375, 275);
39
40                 SplashKit.FillRectangle(Color.Orange, hardButton);
41                 SplashKit.DrawText("Hard", Color.Black, 385, 335);
42
43                 SplashKit.FillRectangle(Color.Red, extremeButton);
44                 SplashKit.DrawText("Extreme", Color.Black, 375, 395);
```

```
45
46                 SplashKit.RefreshScreen(60);
47
48                 // Handle Mouse Click on Difficulty Buttons
49                 if (SplashKit.MouseClicked(MouseButton.LeftButton))
50                 {
51                     Point2D mousePosition = SplashKit.MousePosition();
52
53                     if (SplashKit.PointInRectangle(mousePosition,
                       easyButton))
54                     {
55                         return "Easy";
56                     }
57                     else if (SplashKit.PointInRectangle(mousePosition,
                       mediumButton))
58                     {
59                         return "Medium";
60                     }
61                     else if (SplashKit.PointInRectangle(mousePosition,
                       hardButton))
62                     {
63                         return "Hard";
64                     }
65                     else if (SplashKit.PointInRectangle(mousePosition,
                       extremeButton))
66                     {
67                         return "Extreme";
68                     }
69                 }
70             }
71         }
72
73         public string StartBattle(Character selectedCharacter)
74         {
75             string difficulty = GetSelectedDifficulty(); // Allow the
                 player to choose the difficulty for the battle
76             BattleSystem battleSystem = new BattleSystem(_window);
77             (string battleResult, int reward) = battleSystem.BattleState
                 (selectedCharacter, difficulty);
78             _player.AddZeni(reward);
79             return battleResult;
80         }
81     }
82 }
83
```

```
 1  using SplashKitSDK;
 2  using System;
 3  using DragonBallGame.CharacterClass;
 4
 5  namespace DragonBallGame
 6  {
 7      public class BattleSystem
 8      {
 9          private Random _random = new Random();
10          private Character _player;
11          private Character _villain;
12          private BattleUI _battleUI;
13          private MessageManager _messageManager;
14          private VillainAI _villainAI;
15
16          public BattleSystem(Window window)
17          {
18              _battleUI = new BattleUI(window, new ImageManager());
19              _messageManager = new MessageManager();
20              _villainAI = new VillainAI();
21          }
22
23          public (string, int) BattleState(Character player, string
              difficulty)
24          {
25              _player = player;
26              _villain = GetVillainByDifficulty(difficulty);
27
28              _messageManager.AddMessage($"Battle Started: {_player.Name} vs
                  {_villain.Name} ({difficulty} Difficulty)");
29
30              while (_player.Health > 0 && _villain.Health > 0 && !
                  _battleUI.Window.CloseRequested)
31              {
32                  SplashKit.ProcessEvents();
33
34                  // Escape to Menu by pressing E
35                  if (SplashKit.KeyTyped(KeyCode.EKey))
36                  {
37                      return ($"{_player.Name} has run away!", 0);
38                  }
39
40                  SplashKit.ClearScreen(Color.White);
41
42                  _battleUI.DrawBattleState(_player, _villain);
43                  _messageManager.DrawMessages();
44
45                  ExecuteTurn();
46
```

```
47                  SplashKit.RefreshScreen(60);
48              }
49
50              string battleResult;
51              int reward = 0;
52
53              if (_villain.Health <= 0)
54              {
55                  battleResult = $"{_villain.Name} has been defeated!";
56                  reward = GetRewardByDifficulty(difficulty);
57                  _player.ResetHealth();
58                  _player.ResetEnergy();
59              }
60              else
61              {
62                  battleResult = $"{_player.Name} has been defeated...";
63                  _player.ResetHealth();
64                  _player.ResetEnergy();
65              }
66
67              return (battleResult, reward);
68          }
69
70          private Character GetVillainByDifficulty(string difficulty)
71          {
72              return difficulty switch
73              {
74                  "Easy" => CharacterFactory.CreateCharacter("Frieza"),
75                  "Medium" => CharacterFactory.CreateCharacter("Cell"),
76                  "Hard" => CharacterFactory.CreateCharacter("Buu"),
77                  "Extreme" => CharacterFactory.CreateCharacter("Black"),
78                  _ => CharacterFactory.CreateCharacter("Frieza")
79              };
80          }
81
82          private void ExecuteTurn()
83          {
84              if (_player.Health > 0)
85              {
86                  if (SplashKit.KeyTyped(KeyCode.BKey))
87                  {
88                      HandlePlayerBlock();
89                  }
90                  else if (SplashKit.KeyTyped(KeyCode.SpaceKey))
91                  {
92                      HandlePlayerAttack();
93                  }
94              }
95          }
```

```
 96
 97          private void HandlePlayerBlock()
 98          {
 99              _player.Block();
100              _messageManager.AddMessage($"{_player.Name} is blocking.");
101              _player.IncreaseEnergy(_random.Next(10, 30));
102
103              RefreshBattleUIWithDelay();
104              VillainTurn();
105          }
106
107          private void HandlePlayerAttack()
108          {
109              _player.StopBlocking();
110
111              int damage = (_player.Energy == 100) ? _player.Power / 10 :
                   _random.Next(20, _player.Power / 12);
112
113              if (_player.Energy == 100)
114              {
115                  _messageManager.AddMessage($"{_player.Name} used
                       {_player.SpecialAbility} for {damage} damage!");
116                  _player.ResetEnergy();
117              }
118              else
119              {
120                  _messageManager.AddMessage($"{_player.Name} attacks
                       {_villain.Name} for {damage} damage.");
121                  _player.IncreaseEnergy(_random.Next(10, 30));
122              }
123
124              if (_villain.IsBlocking)
125              {
126                  damage /= 2;
127                  _messageManager.AddMessage($"{_villain.Name} blocked and
                       reduced the damage by half!");
128              }
129
130              _villain.Health -= damage;
131              _villain.Health = Math.Max(_villain.Health, 0);
132
133              RefreshBattleUIWithDelay();
134              _villain.StopBlocking();
135              VillainTurn();
136          }
137
138          private void VillainTurn()
139          {
140              if (_villain.Health > 0)
```

```
141             {
142                 VillainAction action = _villainAI.Action(_villain,
                      _player);
143
144                 if (action == VillainAction.Block)
145                 {
146                     _villain.Block();
147                     _messageManager.AddMessage($"{_villain.Name} is
                       blocking.");
148                     _villain.IncreaseEnergy(_random.Next(10, 30));
149                 }
150                 else if (action == VillainAction.Attack)
151                 {
152                     HandleVillainAttack();
153                 }
154
155                 RefreshBattleUIWithDelay();
156                 _player.StopBlocking();
157             }
158         }
159
160         private void HandleVillainAttack()
161         {
162             int damage = (_villain.Energy == 100) ? _villain.Power / 10 :
                  _random.Next(20, _villain.Power / 12);
163
164             if (_villain.Energy == 100)
165             {
166                 _messageManager.AddMessage($"{_villain.Name} used
                      {_villain.SpecialAbility} for {damage} damage!");
167                 _villain.ResetEnergy();
168             }
169             else
170             {
171                 _messageManager.AddMessage($"{_villain.Name} attacks
                      {_player.Name} for {damage} damage.");
172                 _villain.IncreaseEnergy(_random.Next(10, 30));
173             }
174
175             if (_player.IsBlocking)
176             {
177                 damage /= 2;
178                 _messageManager.AddMessage($"{_player.Name} blocked and
                      reduced the damage by half!");
179             }
180
181             _player.Health -= damage;
182             _player.Health = Math.Max(_player.Health, 0);
183         }
```

```
184
185         private void RefreshBattleUIWithDelay()
186         {
187             SplashKit.ClearScreen(Color.White);
188             _battleUI.DrawBattleState(_player, _villain);
189             _messageManager.DrawMessages();
190             SplashKit.RefreshScreen(60);
191             System.Threading.Thread.Sleep(500);
192         }
193
194         private int GetRewardByDifficulty(string difficulty)
195         {
196             return difficulty switch
197             {
198                 "Easy" => 100,
199                 "Medium" => 300,
200                 "Hard" => 500,
201                 "Extreme" => 1000,
202                 _ => 0
203             };
204         }
205     }
206 }
```

```
1  using SplashKitSDK;
2  using DragonBallGame.CharacterClass;
3
4  namespace DragonBallGame
5  {
6      public class BattleUI
7      {
8          private Window _window;
9          private ImageManager _imageManager;
10
11         public Window Window => _window;
12
13         public BattleUI(Window window, ImageManager imageManager)
14         {
15             _window = window;
16             _imageManager = imageManager;
17         }
18
19         public void DrawBattleState(Character player, Character villain)
20         {
21             // Draw Character Images
22             Bitmap playerImage = _imageManager.GetCharacterImage(player);
23             Bitmap villainImage = _imageManager.GetCharacterImage(villain);
24
25             SplashKit.DrawBitmap(playerImage, 50, 130); // Player image on
                 the left
26             SplashKit.DrawBitmap(villainImage, _window.Width - 250,
                 150); // Villain image on the right
27
28             // Draw Health Bars and Energy Bars
29             DrawHealthBar(50, 80, player.Health, player.MaxHealth,
                 Color.Green, $"{player.Name}");
30             DrawEnergyBar(50, 110, player.Energy, 100, Color.Cyan);
31
32             DrawHealthBar(_window.Width - 250, 80, villain.Health,
                 villain.MaxHealth, Color.Red, $"{villain.Name}");
33             DrawEnergyBar(_window.Width - 250, 110, villain.Energy, 100,
                 Color.Cyan);
34
35             SplashKit.DrawText("Press SPACE to attack, B to block, E to
                 escape", Color.Red, 220, 30);
36         }
37
38         private void DrawHealthBar(int x, int y, int currentHealth, int
             maxHealth, Color barColor, string name)
39         {
40             int barWidth = 200;
41             int barHeight = 20;
42             double healthPercentage = (double)currentHealth / maxHealth;
```

```
43              int currentBarWidth = (int)(barWidth * healthPercentage);
44
45              // Draw health bar background (grey)
46              SplashKit.FillRectangle(Color.Gray, x, y, barWidth, barHeight);
47
48              // Draw current health (colored bar)
49              SplashKit.FillRectangle(barColor, x, y, currentBarWidth,
                  barHeight);
50
51              // Draw character name above health bar
52              SplashKit.DrawText(name, Color.Black, x, y - 20);
53          }
54
55          private void DrawEnergyBar(int x, int y, int currentEnergy, int
              maxEnergy, Color barColor)
56          {
57              int barWidth = 200;
58              int barHeight = 10;
59              double energyPercentage = (double)currentEnergy / maxEnergy;
60              int currentBarWidth = (int)(barWidth * energyPercentage);
61
62              // Draw energy bar background (grey)
63              SplashKit.FillRectangle(Color.Gray, x, y, barWidth, barHeight);
64
65              // Draw current energy (colored bar)
66              SplashKit.FillRectangle(barColor, x, y, currentBarWidth,
                  barHeight);
67          }
68      }
69  }
```

```
using DragonBallGame.CharacterClass;

namespace DragonBallGame
{
    public class VillainAI
    {
        private IVillainStrategy _strategy;

        public VillainAI()
        {
            _strategy = new DefaultVillainStrategy();
        }

        public VillainAction Action(Character villain, Character player)
        {
            SetStrategy(villain, player);
            return _strategy.ChooseAction(villain, player);
        }

        private void SetStrategy(Character villain, Character player)
        {
            int powerDifference = player.Power - villain.Power;

            if (Math.Abs(powerDifference) <= 200)
            {
                _strategy = new DefaultVillainStrategy();
            }
            else if (powerDifference >= 200)
            {
                _strategy = new DefensiveVillainStrategy();
            }
            else
            {
                _strategy = new AggressiveVillainStrategy();
            }
        }
    }


    public enum VillainAction
    {
        Block,
        Attack
    }
}
```

```cs
using DragonBallGame.CharacterClass;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DragonBallGame
{
    public interface IVillainStrategy
    {
        VillainAction ChooseAction(Character villain, Character player);
    }
}

```

```
1  using DragonBallGame.CharacterClass;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace DragonBallGame
9  {
10     public class DefaultVillainStrategy : IVillainStrategy
11     {
12         private Random _random = new Random();
13
14         public VillainAction ChooseAction(Character villain, Character
             player)
15         {
16             double blockChance = _random.NextDouble();
17
18             // If villain's health is below 30% and player's health is more
                than 30%, there's a 60% chance they will block, or simple
               just 20% of random block
19             if ((villain.Health < (villain.MaxHealth * 0.3) && blockChance
               < 0.6 && player.Health > (player.MaxHealth * 0.3)) ||
               blockChance < 0.2)
20             {
21                 return VillainAction.Block;
22             }
23             else
24             {
25                 return VillainAction.Attack;
26             }
27         }
28     }
29 }
30
```

```
1  using DragonBallGame.CharacterClass;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace DragonBallGame
9  {
10     public class AggressiveVillainStrategy : IVillainStrategy
11     {
12         private Random _random = new Random();
13
14         public VillainAction ChooseAction(Character villain, Character
             player)
15         {
16             double blockChance = _random.NextDouble();
17
18             // If villain's health is below 20% and player's health is more
                than 20%, there's a 30% chance they will block, or simple
               just 10% of random block
19             if ((villain.Health < (villain.MaxHealth * 0.2) && blockChance
               < 0.3 && player.Health > (player.MaxHealth * 0.2)) ||
               blockChance < 0.1)
20             {
21                 return VillainAction.Block;
22             }
23             else
24             {
25                 return VillainAction.Attack;
26             }
27         }
28     }
29 }
30
```

```
1  using DragonBallGame.CharacterClass;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace DragonBallGame
9  {
10     public class DefensiveVillainStrategy : IVillainStrategy
11     {
12         private Random _random = new Random();
13
14         public VillainAction ChooseAction(Character villain, Character
             player)
15         {
16             double blockChance = _random.NextDouble();
17
18             // If villain's health is below 50% and player's health is more
                than 30%, there's a 70% chance they will block, or simple
               just 40% of random block
19             if ((villain.Health < (villain.MaxHealth * 0.5) && blockChance
               < 0.7 && player.Health > (player.MaxHealth * 0.3)) ||
               blockChance < 0.4)
20             {
21                 return VillainAction.Block;
22             }
23             else
24             {
25                 return VillainAction.Attack;
26             }
27         }
28     }
29 }
30
```

```
1  using SplashKitSDK;
2  using System;
3  using System.Collections.Generic;
4
5  namespace DragonBallGame
6  {
7      public class MessageManager
8      {
9          private List<(string message, DateTime time)> _messages; // Store
             messages with their timestamp
10         private const double DURATION = 3.0; // Duration to display
             messages in seconds
11         private Font _font;
12
13         public MessageManager()
14         {
15             _messages = new List<(string, DateTime)>();
16             _font = SplashKit.LoadFont("Arial", "Font/arial.ttf");
17         }
18
19         public void AddMessage(string message)
20         {
21             _messages.Clear();
22             _messages.Add((message, DateTime.Now));
23         }
24
25         public void DrawMessages()
26         {
27             List<(string message, DateTime time)> expiredMessages = new
                 List<(string, DateTime)>();
28
29             foreach (var (message, time) in _messages)
30             {
31                 double elapsedSeconds = (DateTime.Now - time).TotalSeconds;
32
33                 if (elapsedSeconds > DURATION)
34                 {
35                     expiredMessages.Add((message, time));
36                 }
37                 else
38                 {
39                     int textWidth = SplashKit.TextWidth(message, _font,
                       14);
40
41                     // Calculate the x-coordinate for center alignment
42                     int xCoordinate = (800 - textWidth) / 2;
43
44                     // Draw the message centered horizontally using the
                       loaded font
```

```
45                      SplashKit.DrawText(message, Color.Red, _font, 14,
                        xCoordinate, 525);
46                  }
47              }
48
49              // Remove expired messages from the list
50              foreach (var expiredMessage in expiredMessages)
51              {
52                  _messages.Remove(expiredMessage);
53              }
54          }
55      }
56  }
57
58
```

```
using SplashKitSDK;
using System.Collections.Generic;
using DragonBallGame.CharacterClass;

namespace DragonBallGame
{
    public class ImageManager
    {
        private Dictionary<string, Dictionary<int, Bitmap>> _characterImages;

        public ImageManager()
        {
            _characterImages = new Dictionary<string, Dictionary<int, Bitmap>>
            {
                { "Son Goku", new Dictionary<int, Bitmap>
                    {
                        { 0, SplashKit.LoadBitmap("Goku_Base", "Resources/goku.png") },
                        { 1, SplashKit.LoadBitmap("Goku_SSJ1", "Resources/goku_ss1.png") },
                        { 2, SplashKit.LoadBitmap("Goku_SSJ2", "Resources/goku_ss2.png") },
                        { 3, SplashKit.LoadBitmap("Goku_SSJ3", "Resources/goku_ss3.png") },
                        { 4, SplashKit.LoadBitmap("Goku_God", "Resources/goku_god.png") },
                        { 5, SplashKit.LoadBitmap("Goku_Blue", "Resources/goku_blue.png") },
                        { 6, SplashKit.LoadBitmap("Goku_UI", "Resources/goku_ultrainstinct.png") }
                    }
                },
                { "Vegeta", new Dictionary<int, Bitmap>
                    {
                        { 0, SplashKit.LoadBitmap("Vegeta_Base", "Resources/vegeta.png") },
                        { 1, SplashKit.LoadBitmap("Vegeta_SSJ1", "Resources/vegeta_ss1.png") },
                        { 2, SplashKit.LoadBitmap("Vegeta_SSJ2", "Resources/vegeta_ss2.png") },
                        { 3, SplashKit.LoadBitmap("Vegeta_God", "Resources/vegeta_god.png") },
                        { 4, SplashKit.LoadBitmap("Vegeta_Blue", "Resources/vegeta_blue.png") },
                        { 5, SplashKit.LoadBitmap("Vegeta_UE", "Resources/vegeta_ultraego.png") }
                    }
```

```
35                  },
36                  { "Son Gohan", new Dictionary<int, Bitmap>
37                      {
38                          { 0, SplashKit.LoadBitmap("Gohan_Base", "Resources/gohan.png") },
39                          { 1, SplashKit.LoadBitmap("Gohan_SSJ1", "Resources/gohan_ss1.png") },
40                          { 2, SplashKit.LoadBitmap("Gohan_SSJ2", "Resources/gohan_ss2.png") },
41                          { 3, SplashKit.LoadBitmap("Gohan_Ultimate", "Resources/gohan_ultimate.png") },
42                          { 4, SplashKit.LoadBitmap("Gohan_Beast", "Resources/gohan_beast.png") }
43                      }
44                  },
45                  { "Frieza", new Dictionary<int, Bitmap>
46                      {
47                          { 0, SplashKit.LoadBitmap("Frieza", "Resources/frieza.png") }
48                      }
49                  },
50                  { "Cell", new Dictionary<int, Bitmap>
51                      {
52                          { 0, SplashKit.LoadBitmap("Cell", "Resources/cell.png") }
53                      }
54                  },
55                  { "Buu", new Dictionary<int, Bitmap>
56                      {
57                          { 0, SplashKit.LoadBitmap("Buu", "Resources/buu.png") }
58                      }
59                  },
60                  { "Black", new Dictionary<int, Bitmap>
61                      {
62                          { 0, SplashKit.LoadBitmap("Black", "Resources/black.png") }
63                      }
64                  }
65              };
66          }
67
68          public Bitmap GetCharacterImage(Character character)
69          {
70              if (_characterImages.ContainsKey(character.Name) &&
71                  _characterImages[character.Name].ContainsKey(character.TransformationLevel))
72              {
73                  return _characterImages[character.Name]
```

```
                    [character.TransformationLevel];
            }

            // Return a default image if no specific image is found
            return SplashKit.LoadBitmap("Default", "Resources/
              default.png");
        }
    }
}

```

```
1  using SplashKitSDK;
2
3  namespace DragonBallGame
4  {
5      public class InputHandler
6      {
7          public void HandleMouseClick(Point2D mousePosition, GameMenu menu)
8          {
9              // Handle Arrow Clicks
10             Rectangle leftArrowRect = SplashKit.RectangleFrom(50, 250, 40,
                 40);
11             if (SplashKit.PointInRectangle(mousePosition, leftArrowRect))
12             {
13                 menu.NavigateLeft();
14                 return;
15             }
16
17             Rectangle rightArrowRect = SplashKit.RectangleFrom(710, 250,
                 40, 40);
18             if (SplashKit.PointInRectangle(mousePosition, rightArrowRect))
19             {
20                 menu.NavigateRight();
21                 return;
22             }
23
24             // Handle Recruit Button Click
25             Rectangle recruitButtonRect = SplashKit.RectangleFrom(250, 550,
                  100, 40);
26             if (SplashKit.PointInRectangle(mousePosition,
                 recruitButtonRect))
27             {
28                 menu.RecruitCharacter();
29                 return;
30             }
31
32             // Handle Battle Button Click
33             Rectangle battleButtonRect = SplashKit.RectangleFrom(450, 550,
                 100, 40);
34             if (SplashKit.PointInRectangle(mousePosition,
                 battleButtonRect))
35             {
36                 menu.Battle();
37                 return;
38             }
39         }
40     }
41 }
42
```

```
1  using SplashKitSDK;
2  using System.Collections.Generic;
3  using static DragonBallGame.RecruitSystem;
4  using DragonBallGame.CharacterClass;
5
6  namespace DragonBallGame
7  {
8      public class GameMenu
9      {
10         private Window _menuWindow;
11         private int _currentCharacterIndex;
12         private ImageManager _imageManager;
13         private Player _player;
14         private InputHandler _inputHandler;
15         private BattleManager _battleManager;
16         private MessageManager _messageManager;
17
18         public GameMenu(Player player)
19         {
20             _menuWindow = new Window("Dragon Ball Game", 800, 600);
21             _player = player;
22             _imageManager = new ImageManager();
23             _inputHandler = new InputHandler();
24             _battleManager = new BattleManager(_player, _menuWindow);
25             _messageManager = new MessageManager();
26
27             // Player starts with only Son Goku
28             _player.AddRecruitedCharacter(new Goku());
29
30             _currentCharacterIndex = 0;
31         }
32
33         public void Run()
34         {
35             while (!_menuWindow.CloseRequested)
36             {
37                 SplashKit.ProcessEvents();
38                 SplashKit.ClearScreen(Color.White);
39
40                 DrawCharacterInfo();
41                 _messageManager.DrawMessages();
42
43                 if (SplashKit.MouseClicked(MouseButton.LeftButton))
44                 {
45                     Point2D mousePosition = SplashKit.MousePosition();
46                     _inputHandler.HandleMouseClick(mousePosition, this);
47                 }
48
49                 SplashKit.RefreshScreen(60);
```

```
50              }
51
52              _menuWindow.Close();
53          }
54
55          private void DrawCharacterInfo()
56          {
57              if (_player.RecruitedCharacters.Count > 0)
58              {
59                  Character currentCharacter = _player.RecruitedCharacters
                      [_currentCharacterIndex];
60                  Bitmap characterImage = _imageManager.GetCharacterImage
                      (currentCharacter);
61
62                  // Draw Character Image
63                  SplashKit.DrawBitmap(characterImage, 150, 50);
64
65                  // Draw Character Stats
66                  SplashKit.DrawText($"{currentCharacter.Name}",
                      Color.Black, 450, 150);
67                  SplashKit.DrawText($"Power: {currentCharacter.Power}",
                      Color.Black, 450, 180);
68                  SplashKit.DrawText($"Health: {currentCharacter.Health}",
                      Color.Black, 450, 210);
69                  SplashKit.DrawText($"Evolution: {currentCharacter.Form}",
                      Color.Black, 450, 240);
70                  SplashKit.DrawText($"Skill:
                      {currentCharacter.SpecialAbility}", Color.Black, 450,
                      270);
71              }
72
73              // Draw Arrow Buttons
74              SplashKit.FillRectangle(Color.LightGray, 50, 250, 40, 40); //
                  Left Arrow
75              SplashKit.FillRectangle(Color.LightGray, 710, 250, 40, 40); //
                   Right Arrow
76              SplashKit.DrawText("<", Color.Black, 65, 265);
77              SplashKit.DrawText(">", Color.Black, 725, 265);
78
79              // Draw Recruit and Battle Buttons
80              SplashKit.FillRectangle(Color.LightGray, 250, 550, 100, 40);
81              SplashKit.DrawText("Recruit", Color.Black, 275, 565);
82              SplashKit.FillRectangle(Color.LightGray, 450, 550, 100, 40);
83              SplashKit.DrawText("Battle", Color.Black, 475, 565);
84
85              // Draw Zeni
86              SplashKit.DrawText($"Zeni: {_player.Zeni}", Color.Black, 650,
                  50);
87          }
```

```
88
89          public void NavigateLeft()
90          {
91              _currentCharacterIndex--;
92              if (_currentCharacterIndex < 0) _currentCharacterIndex =
                  _player.RecruitedCharacters.Count - 1;
93          }
94
95          public void NavigateRight()
96          {
97              _currentCharacterIndex++;
98              if (_currentCharacterIndex >=
                  _player.RecruitedCharacters.Count) _currentCharacterIndex =
                  0;
99          }
100
101         public void RecruitCharacter()
102         {
103             RecruitResult result =
                  RecruitSystem.Instance.RecruitRandomCharacter(_player);
104
105             switch (result.Status)
106             {
107                 case RecruitStatus.NotEnoughZeni:
108                     _messageManager.AddMessage("Not enough Zeni.");
109                     break;
110
111                 case RecruitStatus.NoAvailableCharacters:
112                     _messageManager.AddMessage("All characters are at
                        their highest form and cannot be recruited further.");
113                     break;
114
115                 case RecruitStatus.NewCharacterRecruited:
116                     _player.AddRecruitedCharacter(result.Character);
117                     _messageManager.AddMessage($"{result.Character.Name}
                        recruited!");
118                     break;
119
120                 case RecruitStatus.CharacterEvolved:
121                     _messageManager.AddMessage($"{result.Character.Name}
                        has transformed into {result.Character.Form}!");
122                     break;
123             }
124         }
125
126         public void Battle()
127         {
128             if (_player.RecruitedCharacters.Count == 0) return;
129
```

```
130             Character currentCharacter = _player.RecruitedCharacters
                  [_currentCharacterIndex];
131             string battleResult = _battleManager.StartBattle
                  (currentCharacter);
132             _messageManager.AddMessage($"{battleResult}");
133         }
134     }
135 }
136
```

```
1  namespace DragonBallGame
2  {
3      class Program
4      {
5          static void Main(string[] args)
6          {
7              Player player = new Player();
8              GameMenu menu = new GameMenu(player);
9              menu.Run();
10         }
11     }
12 }
13
```