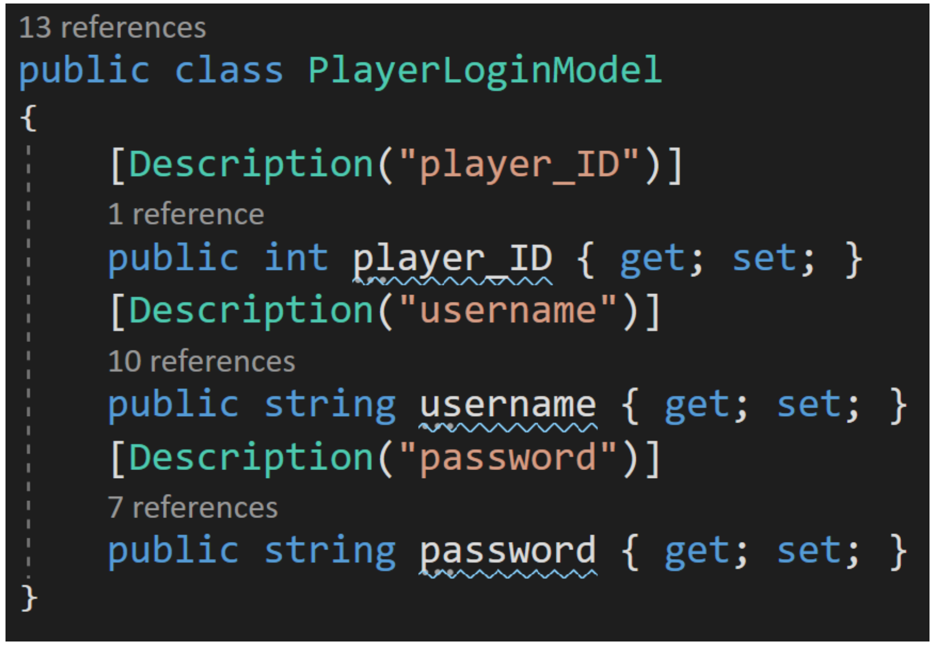
GameSystemObjects Documentation

John Garner, Tyler Stanton, Placidie Mugabo,

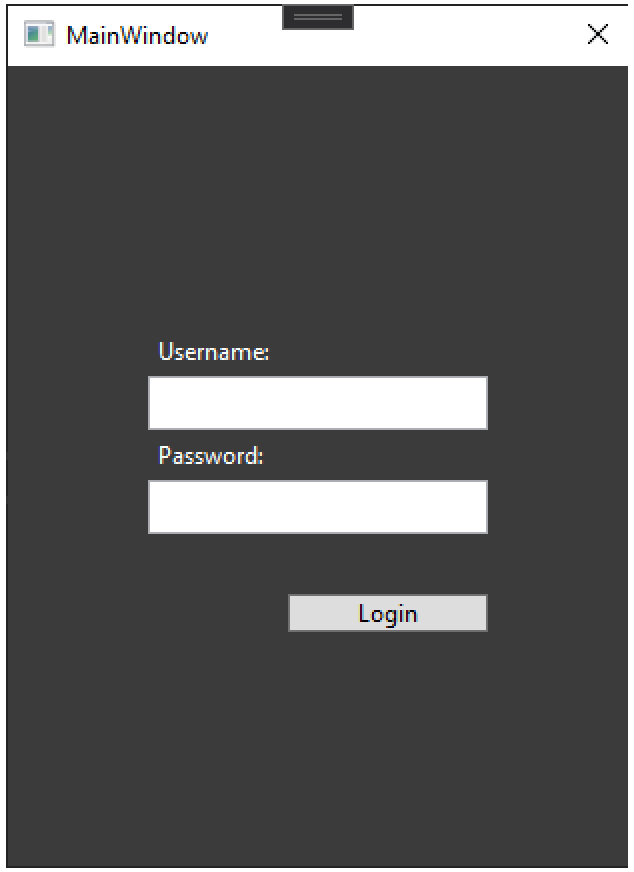
Bailey Costello, Alan Farmer

PlayerLoginModel

- Class to hold data used to login a user.
- Holds the
  - player_ID
  - Username
  - password.



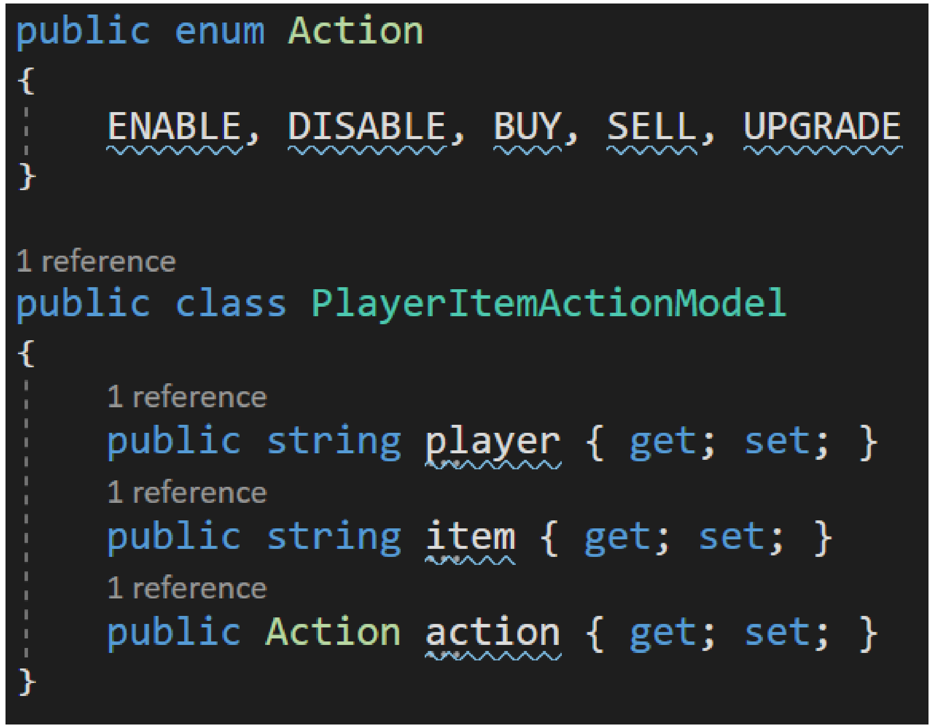
```
13 references
public class PlayerLoginModel
{
    [Description("player_ID")]
    1 reference
    public int player_ID { get; set; }
    [Description("username")]
    10 references
    public string username { get; set; }
    [Description("password")]
    7 references
    public string password { get; set; }
}
```



PlayerItemActionModel

Holds a enumerator and class.

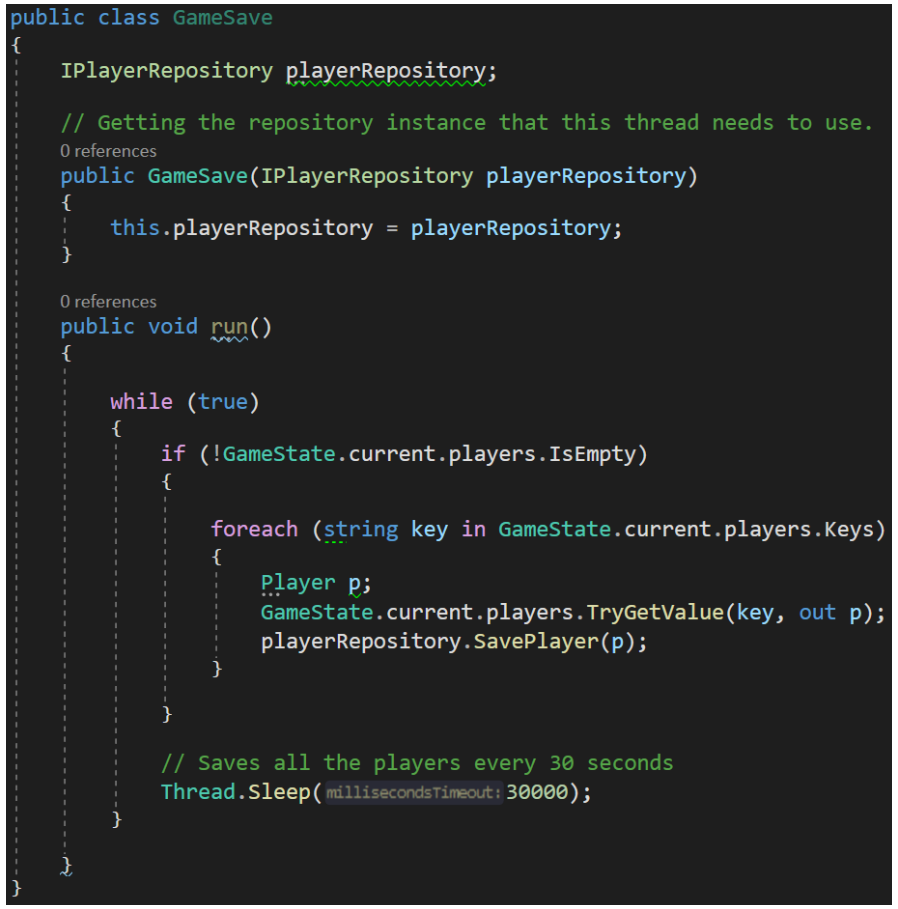
- Class that holds data for communicating actions players take on certain task.
- enum **Action** - Represents possible actions for each task.



```
public enum Action
{
    ENABLE, DISABLE, BUY, SELL, UPGRADE
}

1 reference
public class PlayerItemActionModel
{
    1 reference
    public string player { get; set; }
    1 reference
    public string item { get; set; }
    1 reference
    public Action action { get; set; }
}
```

Game.cs

- Gameloop - Main game loop. While running, checks that there are players and loops through each player to increment all of their items.
- UpdatePlayerGameSpeed - Updates each players time calculation for their current item task.
- GameSave - Loops through each player and saves them to the player repository every 30 seconds.

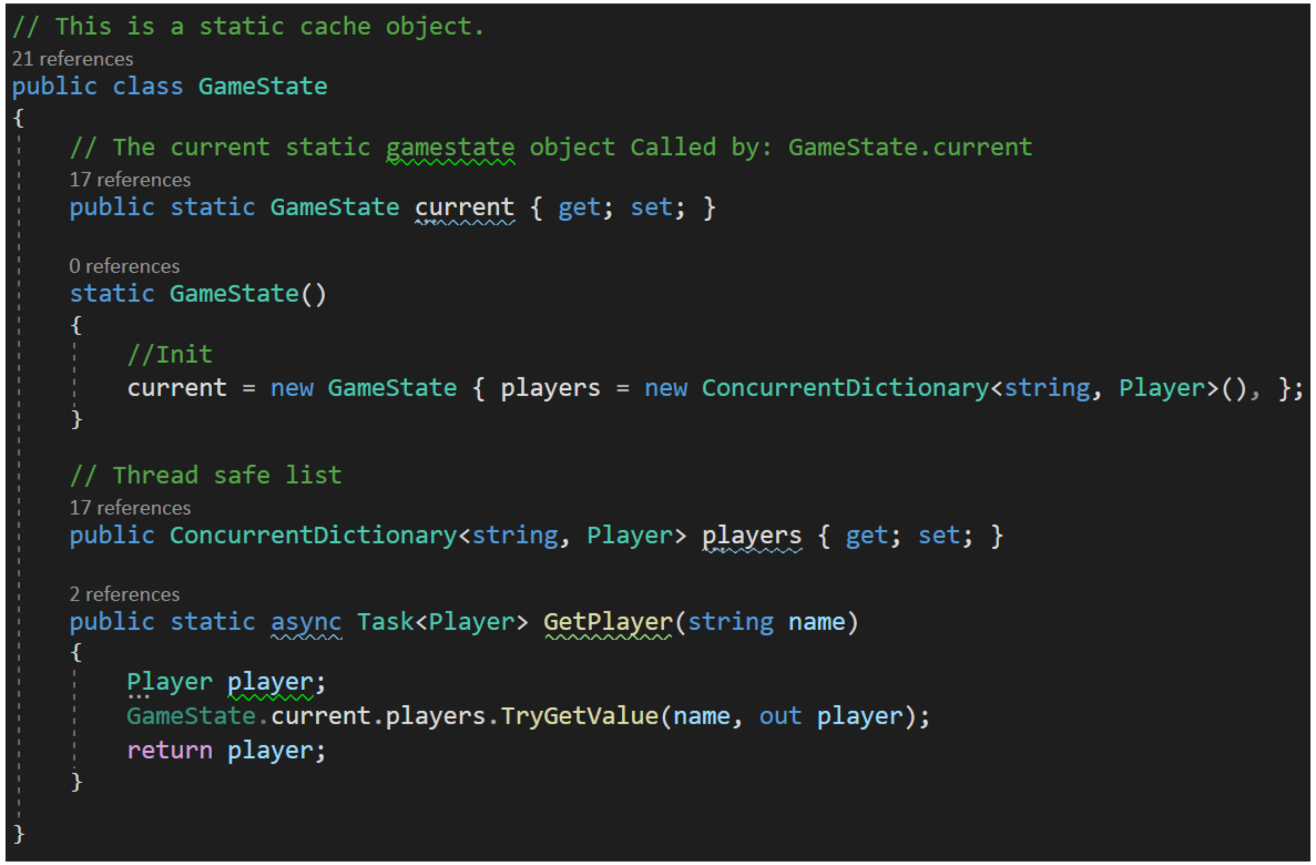


```
public class GameSave
{
    IPlayerRepository playerRepository;

    // Getting the repository instance that this thread needs to use.
    0 references
    public GameSave(IPlayerRepository playerRepository)
    {
        this.playerRepository = playerRepository;
    }

    0 references
    public void run()
    {

        while (true)
        {
            if (!GameState.current.players.IsEmpty)
            {

                foreach (string key in GameState.current.players.Keys)
                {
                    Player p;
                    GameState.current.players.TryGetValue(key, out p);
                    playerRepository.SavePlayer(p);
                }

            }

            // Saves all the players every 30 seconds
            Thread.Sleep(millisecondsTimeout: 30000);
        }

    }
}
```

GameSave Code Example

Game.cs

- CleanUpSessions - Checks through each player to check if more than a minute has passed since their last seen time, if so, saves and trys to remove to player from the current repository.
- Game - Builds both the GameLoop and CleanUpSessions threads.
- GameState - Static cache object with a currnet gamestate and thread safe list.

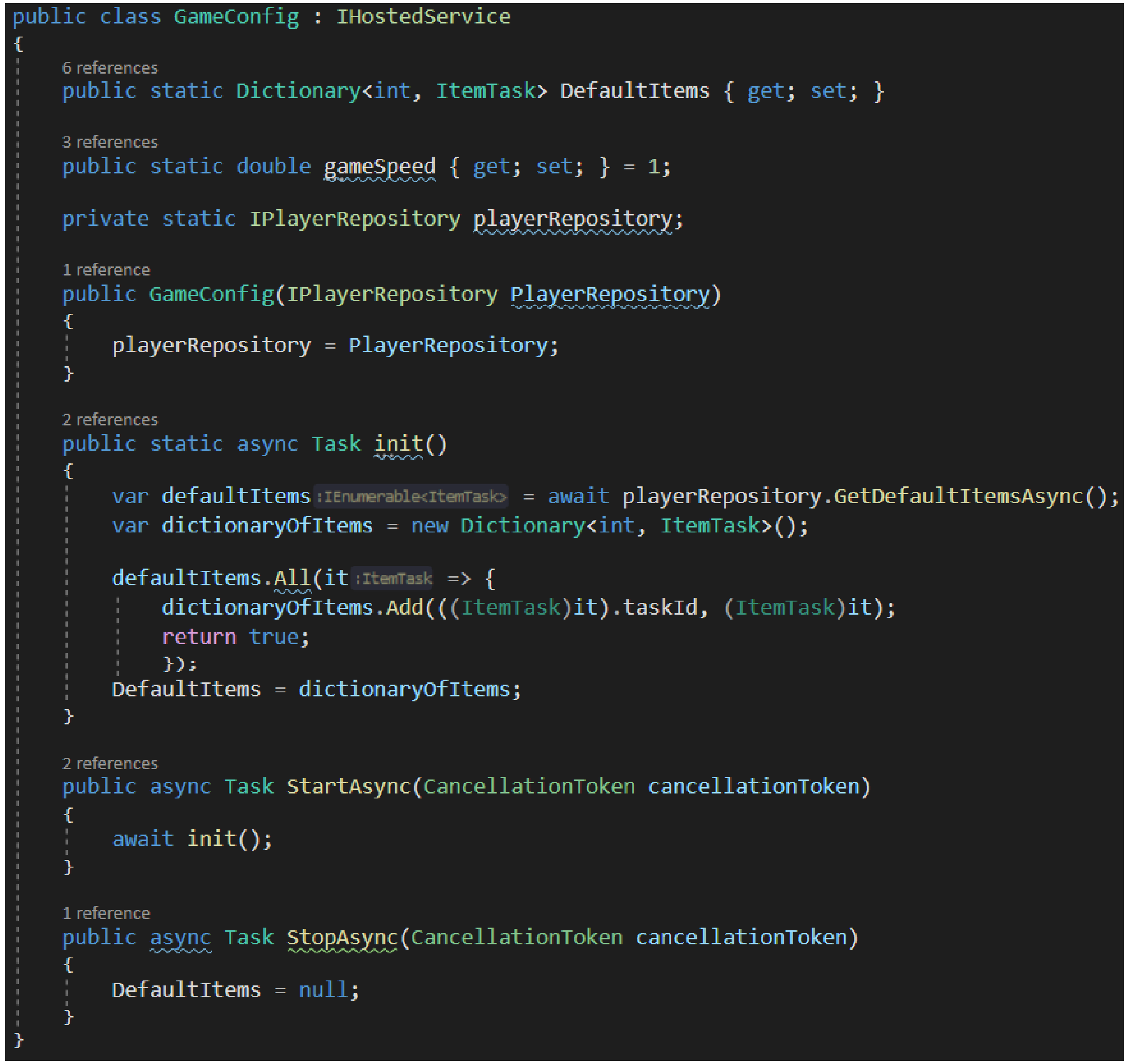


```csharp
// This is a static cache object.
21 references
public class GameState
{
    // The current static gamestate object Called by: GameState.current
    17 references
    public static GameState current { get; set; }

    0 references
    static GameState()
    {
        //Init
        current = new GameState { players = new ConcurrentDictionary<string, Player>(), };
    }

    // Thread safe list
    17 references
    public ConcurrentDictionary<string, Player> players { get; set; }

    2 references
    public static async Task<Player> GetPlayer(string name)
    {
        Player player;
        GameState.current.players.TryGetValue(name, out player);
        return player;
    }

}
```

GameState Example Code

GameConfig.cs

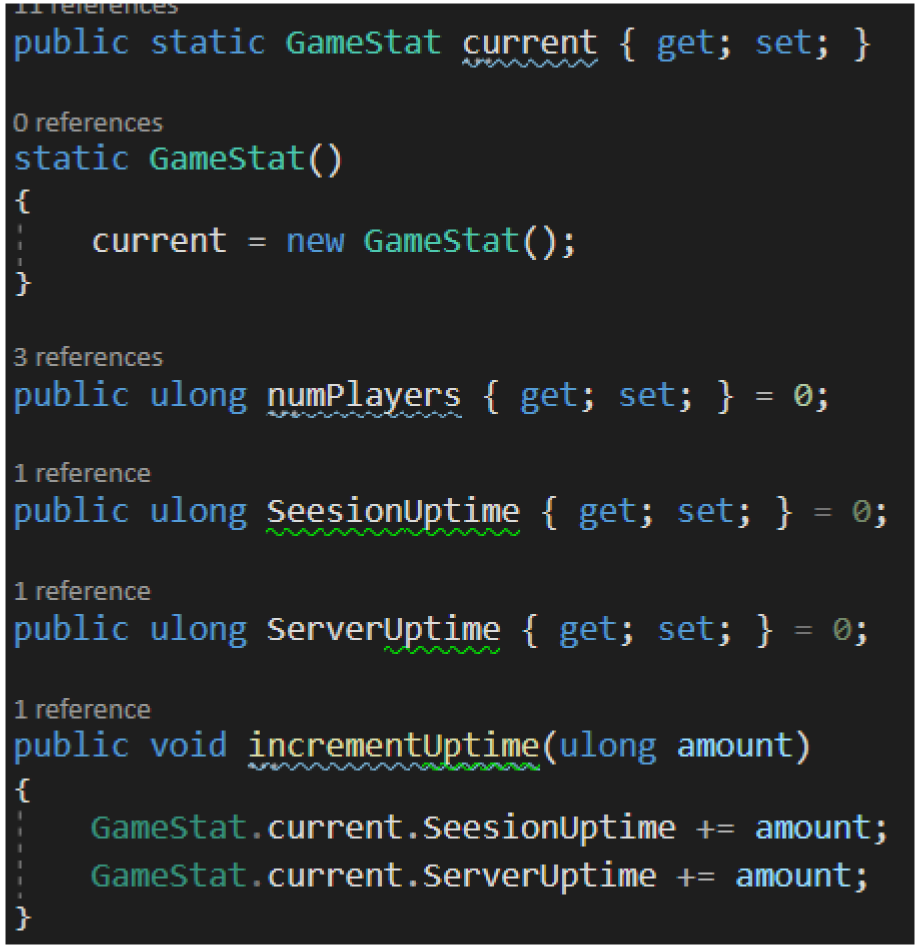
- Defines objects:
  - Dictionary DefaultItems
  - double GameSpeed
  - playerRepository
  - class GameConfig
  - Task init
  - Task StartAsync
  - Task StopAsync



```
public class GameConfig : IHostedService
{
    6 references
    public static Dictionary<int, ItemTask> DefaultItems { get; set; }

    3 references
    public static double gameSpeed { get; set; } = 1;

    private static IPlayerRepository playerRepository;

    1 reference
    public GameConfig(IPlayerRepository PlayerRepository)
    {
        playerRepository = PlayerRepository;
    }

    2 references
    public static async Task init()
    {
        var defaultItems :IEnumerable<ItemTask> = await playerRepository.GetDefaultItemsAsync();
        var dictionaryOfItems = new Dictionary<int, ItemTask>();

        defaultItems.All(it :ItemTask => {
            dictionaryOfItems.Add(((ItemTask)it).taskId, (ItemTask)it);
            return true;
            });
        DefaultItems = dictionaryOfItems;
    }

    2 references
    public async Task StartAsync(CancellationToken cancellationToken)
    {
        await init();
    }

    1 reference
    public async Task StopAsync(CancellationToken cancellationToken)
    {
        DefaultItems = null;
    }
}
```

GameStat.cs

- Defines classes to track game statistics:
  - numPlayer - number of players
  - SessionUpTime - Players time online
  - ServerUpTime - Time the server has been online
  - globalItemTaskStats - Dictionary of ItemStat
  - globalItemTaskLeaderBoard - Key value pair dictionary ot create a leaderboard of players items.

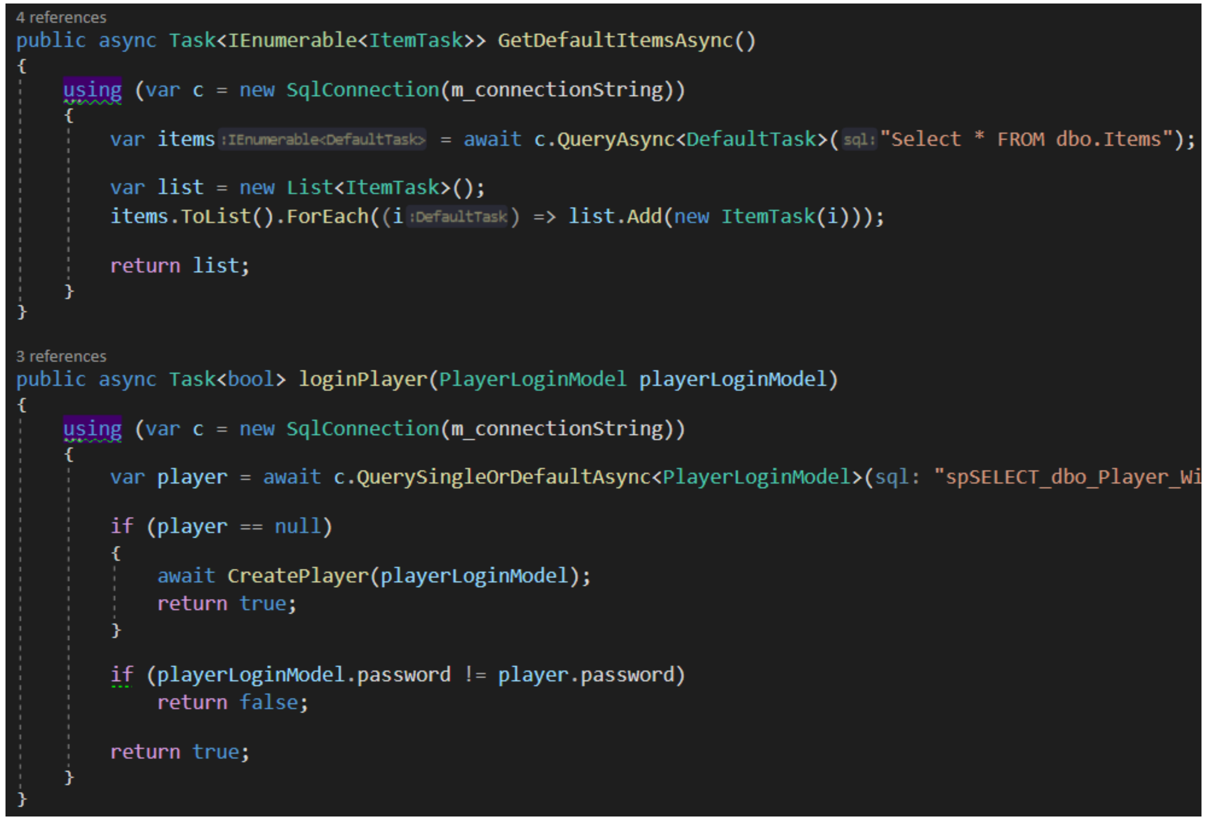


```
public static GameStat current { get; set; }

0 references
static GameStat()
{
    current = new GameStat();
}

3 references
public ulong numPlayers { get; set; } = 0;

1 reference
public ulong SeesionUptime { get; set; } = 0;

1 reference
public ulong ServerUptime { get; set; } = 0;

1 reference
public void incrementUptime(ulong amount)
{
    GameStat.current.SeesionUptime += amount;
    GameStat.current.ServerUptime += amount;
}
```

PlayerRepository.cs

Defines task objects in PlayerRepository

- GetPlayer - Requests players information from the database
- SavePlayer – saves a player
- GetDefaultItemsAsync - Requests default items from the database
- loginPlayer - Requests and compairs login information
- CreatePlayer - Adds a new player to the database
- RemovePlayer - Removes a player from the database
- GetStats - WIP

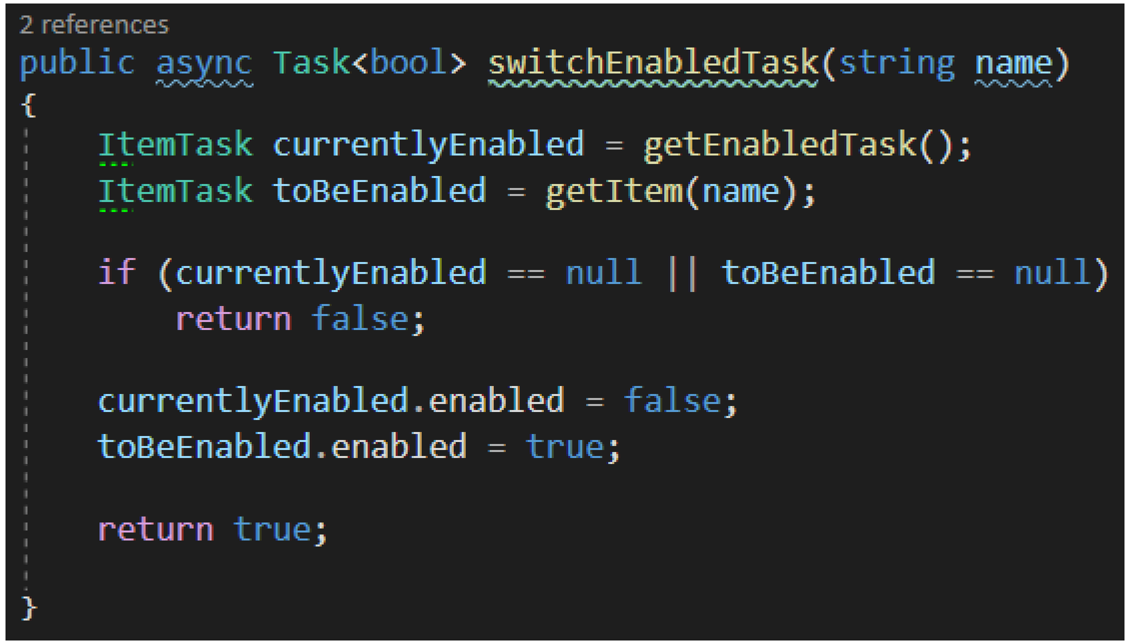


```
4 references
public async Task<IEnumerable<ItemTask>> GetDefaultItemsAsync()
{
    using (var c = new SqlConnection(m_connectionString))
    {
        var items :IEnumerable<DefaultTask> = await c.QueryAsync<DefaultTask>( sql: "Select * FROM dbo.Items");

        var list = new List<ItemTask>();
        items.ToList().ForEach((i :DefaultTask) => list.Add(new ItemTask(i)));

        return list;
    }
}

3 references
public async Task<bool> loginPlayer(PlayerLoginModel playerLoginModel)
{
    using (var c = new SqlConnection(m_connectionString))
    {
        var player = await c.QuerySingleOrDefaultAsync<PlayerLoginModel>(sql: "spSELECT_dbo_Player_Wi

        if (player == null)
        {
            await CreatePlayer(playerLoginModel);
            return true;
        }

        if (playerLoginModel.password != player.password)
            return false;

        return true;
    }
}
```

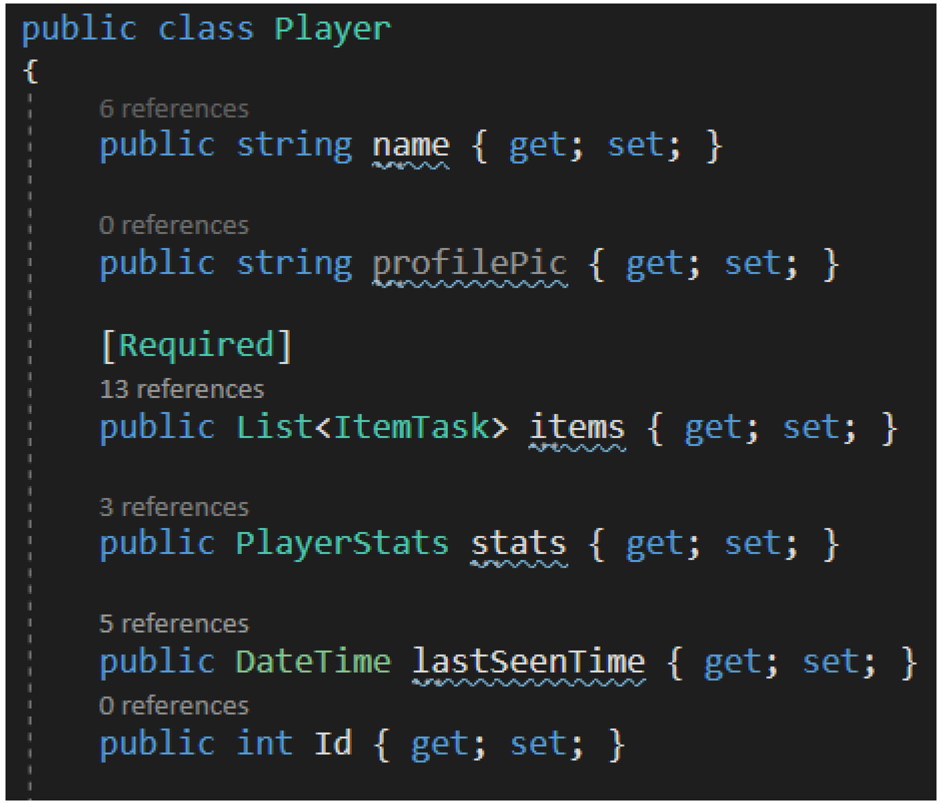
PlayerRepository Example Code

Player.cs

- Class for players information.
- Holds Task for interacting with the itemTasks



```
2 references
public async Task<bool> switchEnabledTask(string name)
{
    ItemTask currentlyEnabled = getEnabledTask();
    ItemTask toBeEnabled = getItem(name);

    if (currentlyEnabled == null || toBeEnabled == null)
        return false;

    currentlyEnabled.enabled = false;
    toBeEnabled.enabled = true;

    return true;

}
```

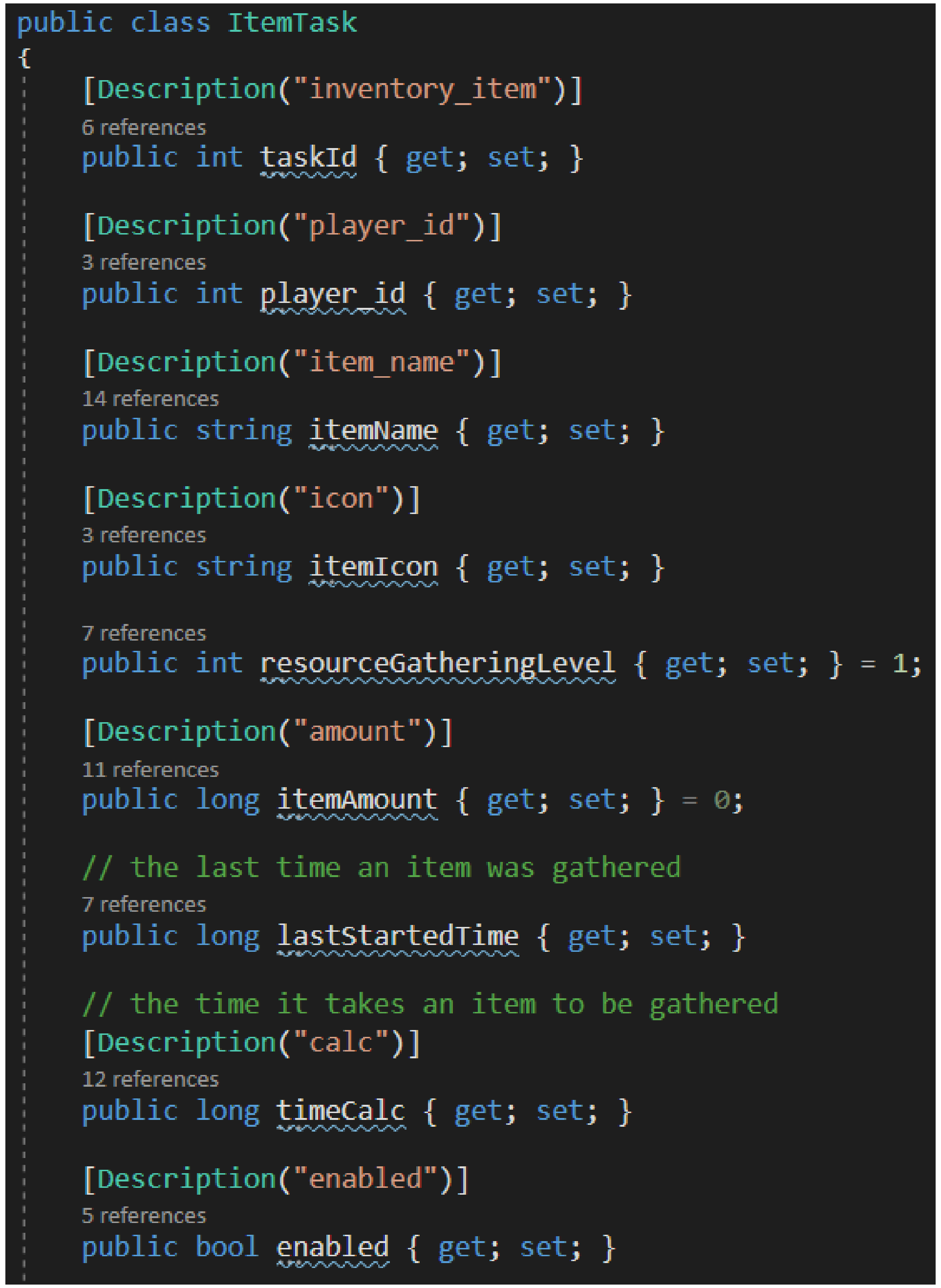


```
public class Player
{
    6 references
    public string name { get; set; }

    0 references
    public string profilePic { get; set; }

    [Required]
    13 references
    public List<ItemTask> items { get; set; }

    3 references
    public PlayerStats stats { get; set; }

    5 references
    public DateTime lastSeenTime { get; set; }
    0 references
    public int Id { get; set; }
```

ItemTask.cs

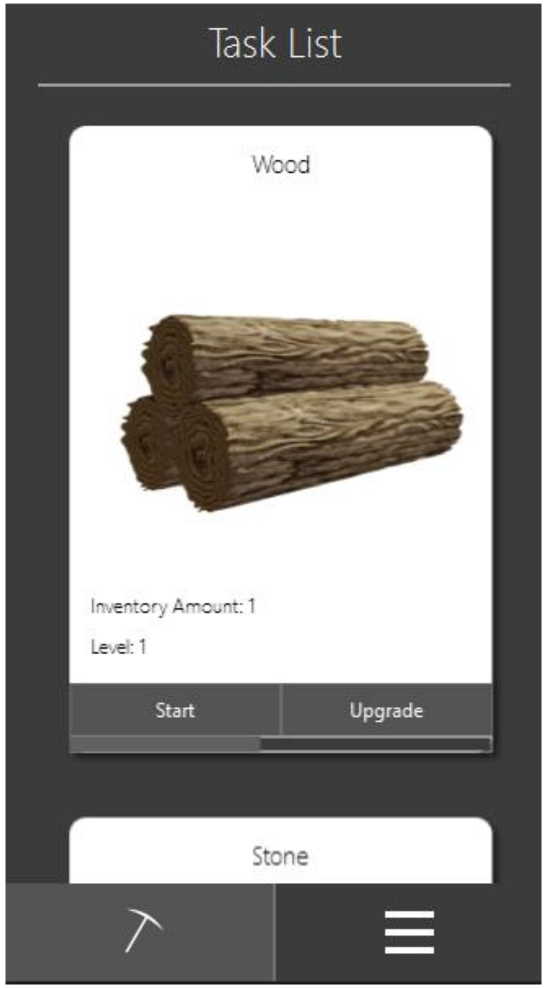
Object that holds all task data.

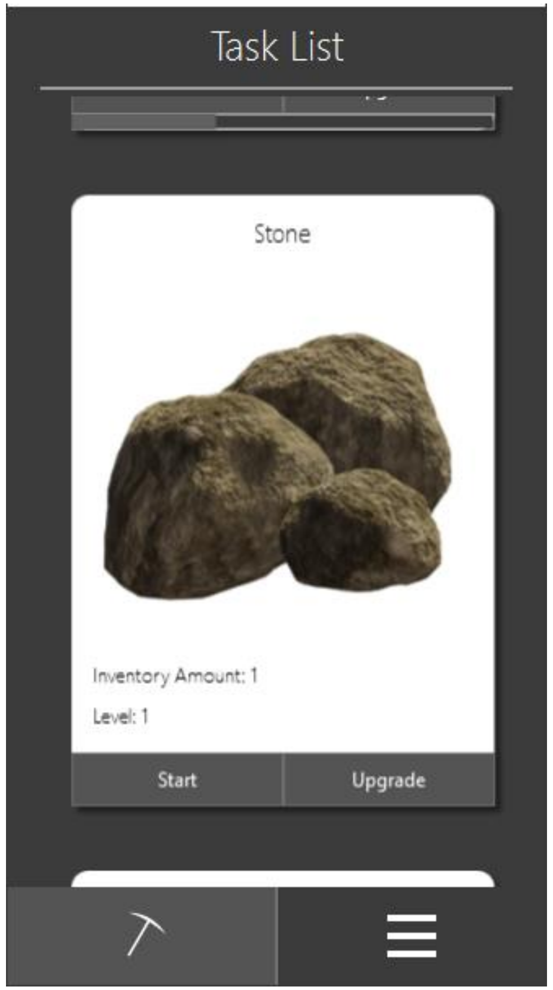
- taskId - Task ID for Database, int
- itemName - Item/Task name, string
- itemIcon - Location for Item/Task Icon, string
- resourceGatheringLevel - The upgrade level of the task, int
- itemAmount - Amount of this item this player has, long
- lastStartedTime - The last time an item was gathered, long
- timeCalc - The time it takes for the task to complete and add to itemAmount, long
- enabled - Whether the task is running or not, bool
- upgradeGatheringLevelCost - Method to determin the upgrade cost of this task, int

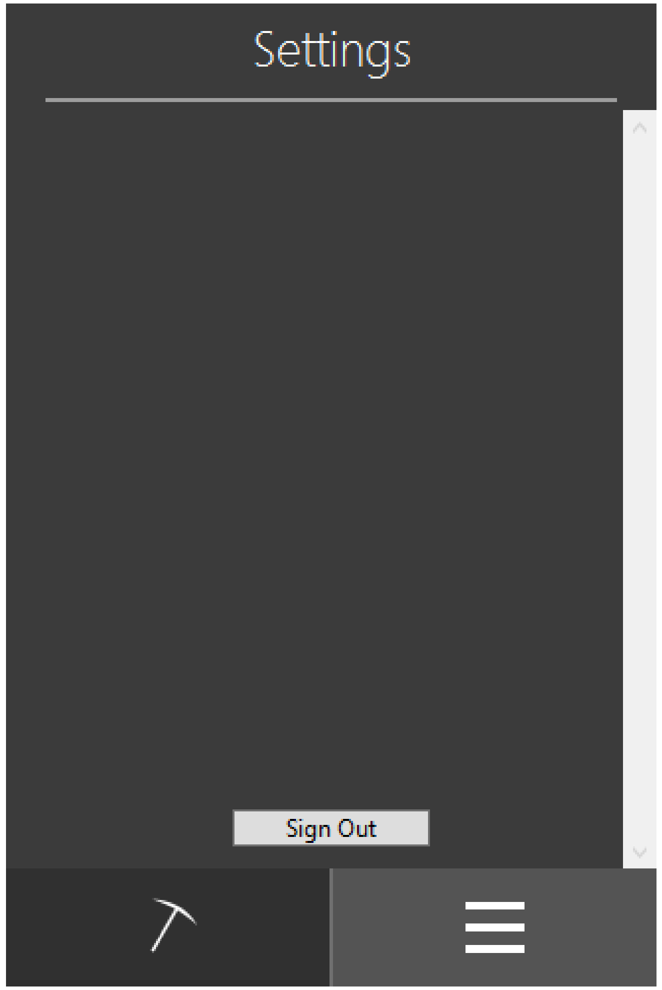


```
public class ItemTask
{
    [Description("inventory_item")]
    6 references
    public int taskId { get; set; }

    [Description("player_id")]
    3 references
    public int player_id { get; set; }

    [Description("item_name")]
    14 references
    public string itemName { get; set; }

    [Description("icon")]
    3 references
    public string itemIcon { get; set; }

    7 references
    public int resourceGatheringLevel { get; set; } = 1;

    [Description("amount")]
    11 references
    public long itemAmount { get; set; } = 0;

    // the last time an item was gathered
    7 references
    public long lastStartedTime { get; set; }

    // the time it takes an item to be gathered
    [Description("calc")]
    12 references
    public long timeCalc { get; set; }

    [Description("enabled")]
    5 references
    public bool enabled { get; set; }
```

App Main Screens While Running:







Settings Not Implemented