Hacettepe University - Computer Engineering

BBM203 Software Laboratory I - Fall 2024

PROGRAMLAMA ÖDEVI 4

Başlıklar: İkili Ağaçlar, Kendini Dengeleyen Ağaçlar,

Dinamik Bellek Tahsisi, Operatör Aşırı Yüklemesi, Dosya G/Ç

Kurs Eğitmenleri: Doç Dr. Adnan ÖZSOY, Yrd Doç Dr. Engin DEMİR, Doç Dr. Hacer YALIM KELEŞ
Doç: M. Aslı TAŞ¸GETİREN* , S. Meryem TAŞ¸YÜREK, Dr. Öğr. Üyesi Selma DİLEK
Programlama Dili: C++11 - **Bu başlangıç kodunu KULLANMALISINIZ**
Son Teslim Tarihi: 15/12/2024 Pazar (23:59:59) https://submit.cs.hacettepe.edu.tr üzerinden

Giriş

M.A.T Game Studios'un (H.U. Game Studios'un bir yan kuruluşu) heyecan verici haberleri var - popüler RPG'leri H.U.SLAND'e yeni bir oyuncu sınıfı eklemeye karar verdiler! Ancak, oyunun yeni versiyonu geliştirilmeden önce, şirket yöneticileri bu yeni fikir için bir kavram kanıtı prototipi görmek istiyor. Bu görev amirleriniz tarafından Hacettepe Üniversitesi'nden siz güvenilir stajyerlere verildi.

Yeni sürümle birlikte H.U.SLAND oyunu artık özel bir sınıfa sahip olacak: **Diyar Şekillendiriciler**. Realm Shapers, oyun dünyasının Adalarını işleyip yeniden şekillendirebilen özel bir maceracı sınıfıdır. Bu sınıfa ulaşabilen oyuncular, oyun dünyasını kendi kaprislerine göre değiştirebilecek ve daha yüksek rütbeler için diğer oyunculara meydan okuyabilecek. Ancak her şeyin bir bedeli vardır ve Realm Shaper kazanması zor ama kaybetmesi kolay bir unvandır, bu da stratejik oynanışı gerekli kılar.

Bu yeni oyun dinamiği hakkında şimdiden söylentiler dolaşıyor ve sıkı hayranlar büyük bir heyecanla bekliyor. Stajyerlerin amirleri onlara sadece iki hafta verdi, yeni özellikleri tamamen ilk kimin uygulayacağını görmek için bir yarış var.

Yeni oyunun işleyişi gizlidir, ancak Hacettepe Üniversitesi'nden güvenilir stajyerlerin sırrı öğrenmesine izin verilmiştir. Bu yeni oyun mekanizmasının arkasındaki sır nedir? Oyun dünyası artık kendi kendini dengeleyen bir ağaç ile çalışacak. Bu sayede hem arka uç kodundaki adalara erişim kolaylaşacak hem de oyun dünyasının kullanıcı arayüzünde bilinmeyenler için tahmin edilemeyecek heyecan verici değişiklikler olacak.

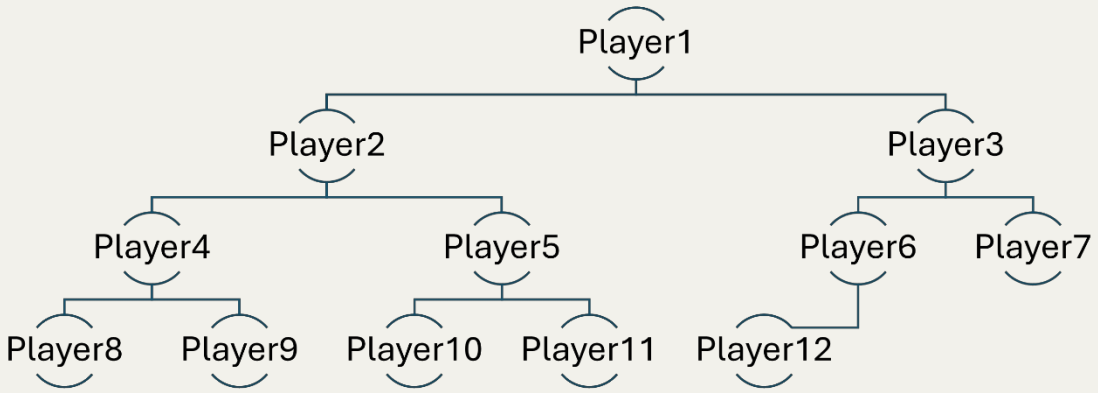
1 Yeni Karakter Sınıfı: Diyar Şekillendiriciler

1.1 Birçoğunun ilki: Şekillendirici Ağaç

Shaper Tree, bir dizi (veya bir vektör) kullanılarak uygulanan özel bir ikili ağaçtır. Bu ağaçta yeni eklenen düğümler her zaman soldan sağa doğru seviye seviye eklenerek eksiksiz bir ikili ağaç yapısı sağlanır.

Silme işlemi, düğümün silinmesi ve diğer düğümlerin boşluğu doldurmak için sıralarını koruyarak kaydırılmasıyla yapılır.

Sıralama sistemi, ağacın seviye-sıra geçişinden türetilmiştir.



Şekil 1: Oyuncu1'i **sırayla** 12'ye ekledikten **sonraki** ağaç

1.2 Tüm Diyar Şekillendiriciler aynı değildir: Sınıf

Diyar Şekillendiricilerin kendi aralarında bir hiyerarşisi vardır. Şekillendirici Ağacındaki konumlarına göre belirlenen Âlem Şekillendiricilerin derecesi, Âlem Şekillendirici unvanları aktifken ziyaret edebilecekleri dünya bölümlerini etkiler. Olabildiğince çok Adayı ziyaret edebilmek Şekillendiriciler için gereklidir çünkü yeni diyarlar yaratmak için gerekli eşyaları ancak bu şekilde toplayabilirler.

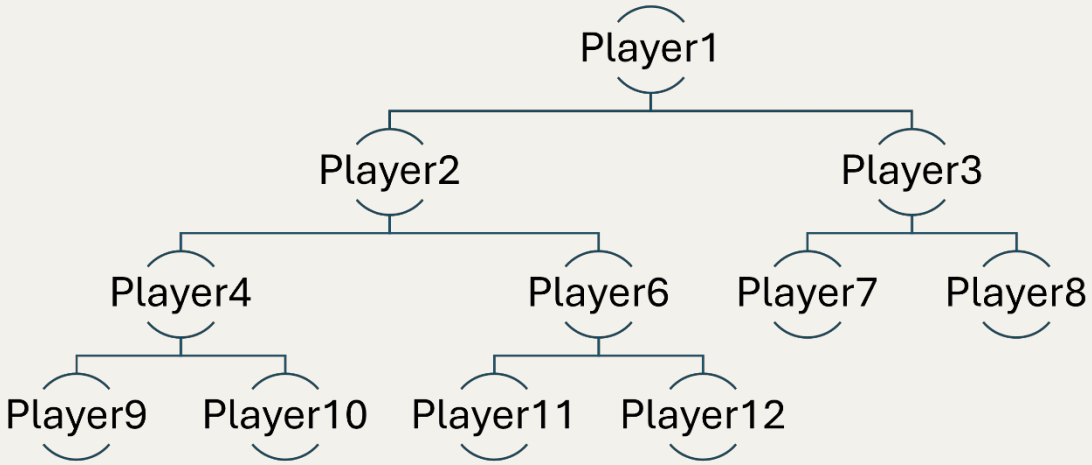
Diyar Şekillendiricilerin Dereceleri, Şekillendirici Ağacındaki derinliklerine göre belirlenir. Köke daha yakın olan oyuncuların Dereceleri daha yüksektir ve daha fazla Adaya erişimleri vardır, bu da yeni diyarlar oluşturmak için gerekli olan değerli eşyaları toplamalarına olanak tanır. Erişim hakkında daha fazla bilgi Bölüm 1.5'te.

1.3 Seviye Atlama Fırsatları: Düellolar

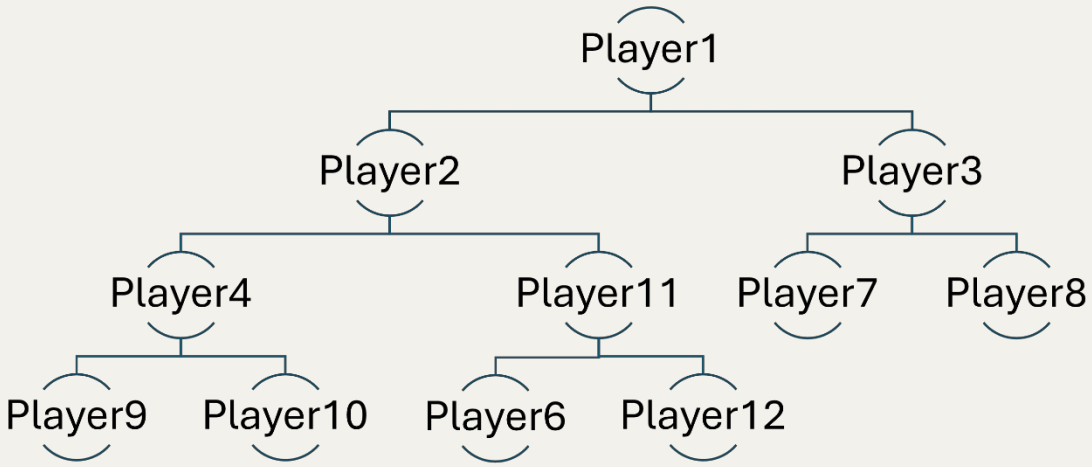
Oyuncular, bağlı oldukları üstün oyuncuyu (ana düğüm) düelloya davet ederek Realm Shaper sıralamalarını geliştirebilirler. Bir düello kazanılırsa, oyuncular birbirleriyle yer değiştirir.

Düellolar ayrıca özel nitelik olan **Onur** puanı kazanmak için de iyi fırsatlardır. Her Diyar Şekillendirici, unvanını kazandığında 1000 Onur Puanı alır. Kazanılan her düello daha fazla Onur Puanı ekler.

oyuncu. Ancak, oyuncular başkalarını düelloya davet ederken dikkatli olmalıdır çünkü düellolardan birini kaybetmek oyuncuya Onur Puanı kaybettirir. Şeref Puanı sıfır olan bir oyuncu Şekillendirici Ağacından elenir ve Diyar Şekillendiricisi unvanını kaybeder.



Şekil 2: Oyuncu5 silindikten sonraki ağaç (Oyuncu2'ye çok fazla düello kaybettiğini varsayalım)



Şekil 3: Oyuncu11 düelloyu kazandıktan sonraki ağaç.

1.4 Her şeyin bir fiyatı vardır: Öğeler

Tüm Diyar Şekillendiriciler yeni diyarlar yaratmak için teknik beceriye sahip olsalar da, gerekli Şekillendirme Enerji Puanlarına sahip olmadan bunu yapamazlar. Bu gerekli enerji puanlarını elde etmek için Şekillendiricilerin öncelikle gerekli malzemeleri toplaması gerekir. Toplanan tüm öğeler işlenerek Şekillendiricilerin kullanabileceği Şekillendirme Enerjisine dönüştürülür. Enerji sağlayan öğeler dünya haritasına dağılmıştır ve yalnızca aktif Diyar Şekillendirici unvanına sahip oyuncular tarafından görülebilir. Öğeler hakkında daha fazla bilgi Bölüm 2.2'de.

1.5 Güçlü ama OP değil: Erişim

Aktif Diyar Şekillendirici unvanı olmayan oyuncuların ne yaptığı bu görevin konusu değildir. Bir Diyar Şekillendiricisinin bir Adaya gerekli erişimi olmadığında, unvanlarını devre dışı bıraktıktan sonra oraya gitmiş olsalar bile orada yok sayılacaklardır. Bölüm 1.2.'de belirtildiği gibi erişim, Şekillendirici Ağacındaki oyuncuların derinliği gibi Dereceye göre belirlenir.

Erişim için üç kural vardır:

1. Kökteki oyuncu her şeye erişebilir.
2. En alttaki oyuncular dünyadaki sadece dış Adalara erişebilir.
3. Ara oyuncuların erişimi hiyerarşideki konumlarıyla orantılıdır. Oyuncunun Dünya Ağacındaki erişim seviyesi (minMapDepthAccess) şu şekilde hesaplanır:

$$\text{minMapDepthAccess} = \text{totalMapDepth} \times \frac{\text{playerDepth}}{\text{toplamYüksekliği}}$$

- Nerede?
  - playerDepth: Oyuncu düğümünün köke olan uzaklığı (köke olan kenar sayısı).
  - totalHeight: Şekillendirici Ağacın toplam yüksekliği.
  - totalMapDepth: Dünya Ağacının (harita) maksimum derinliği.
  - minMapDepthAccess: Bir oyuncunun erişebileceği Dünya Ağacındaki minimum derinlik, örneğin 0 ise oyuncu 0'dan büyük ve 0'a eşit derinliğe sahip düğümlere, yani her yere erişebilir.

1.6 Ne ekersen onu biçersin: FileIO

Diyar Şekillendirici olabilmiş oyuncular bir metin dosyasında tutulur. Bunun için girdi dosyası initial_realm_shapers.txt ve çıktı dosyası current_realm_shapers.txt olacaktır. Bu dosyaların her ikisi de sıralamalarına göre listelenmiş oyuncuları içermelidir. Bununla ilgili daha fazla bilgi Bölüm 4'te.

1.7 Kullanıma Hazır: Terminal Çıkışı

Oyuncular terminale bu formatta yazılır. Başlangıç kodu bu kodu içerir, bunu uygulamayacaksınız.

Terminaldeki Şekillendirici Ağaç:

```
İsim: Oyuncu1
   ---- İsim: Oyuncu2
          ---- İsim: Oyuncu4
                 ---- İsim: Oyuncu8
                 ---- İsim: Oyuncu9
          ---- İsim: Oyuncu5
                 ---- İsim: Oyuncu10
                 ---- İsim: Oyuncu11
   ---- İsim: Oyuncu3
          ---- İsim: Oyuncu6
          ---- İsim: Oyuncu7
```

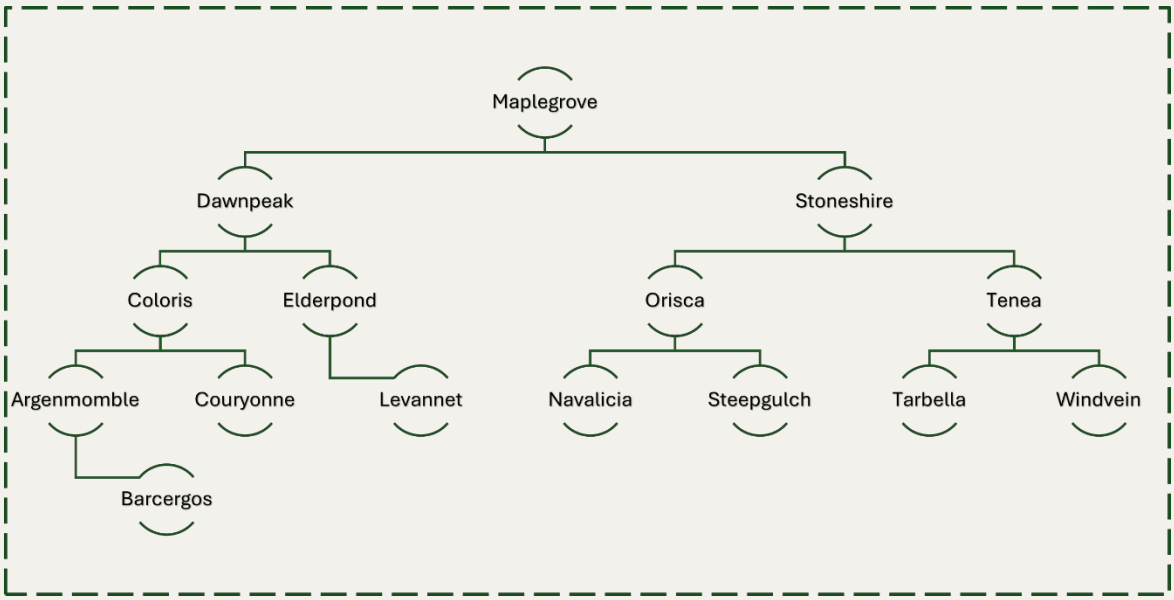
2 Yeni Dünya Düzeni: AVL

Yeni karakter sınıfı sürümünden yararlanan geliştiriciler, aynı zamanda Dünyaları için haritadaki Adalara hem daha tutarlı hem de verimli erişime sahip yeni bir oyun mekanizması sunmaya karar verdiler. Bunun için kendi kendini dengeleyen bir ağaç, özellikle de AVL ağacı kullanmaya karar verdiler. Bu sayede yeni oyun mekanizmaları da ekleyebildiler. Sıradan oyuncular için gizemli ve beklenmedik dünya değişiklikleri ve Realm Shapers için oyun dünyasını kendi avantajlarına göre değiştirme şansı gibi.

Arka uçta, AVL işaretçiler kullanılarak uygulanmaktadır. Ağaçtaki her düğüm, adı ve ilişkili herhangi bir öğe gibi niteliklere sahip bir Isle tutar. İşaretçi tabanlı uygulama, dinamik bellek tahsisini yönetmede esneklik sağlayarak verimli ekleme, silme ve çaprazlama işlemlerine olanak tanır. AVL, dünya dinamik olarak değişse bile Adalara hızlı erişim için optimum yüksekliği koruyarak ağacın rotasyonlar boyunca dengesini korumasını sağlar.

2.1 Dünya Başlatılıyor: Harita

Dünya Ağacı (Harita) initial_world.txt girdi dosyasından başlatılır. AVL dengesi korunarak düğümler (Adalar) teker teker eklenir.



Şekil 4: Tüm adalar eklendikten sonra örnek ağaç

2.2 Çaprazlayın ve Zenginleştirin: Öğeler

Dünya Ağacı (Harita) başlatıldıktan sonra, Realm Shapers'ın işçilikte kullanacağı özel öğelerle doldurulur. Öğeler, her biri benzersiz değerlere sahip Öğeler enumunda kodlanmıştır. Her Ada aynı anda yalnızca bir tür Eşya barındırabilir. Zenginleştirme, Goldium ve Einsteinium için başlatıldıktan sonra bir kez yapılacak ve bundan sonra tüm öğeler için her 3. kez dünyanın yeniden dengelenmesi gerekecektir.

2.2.1 Ürün Mekaniği

Sipariş Sonrası Geçiş

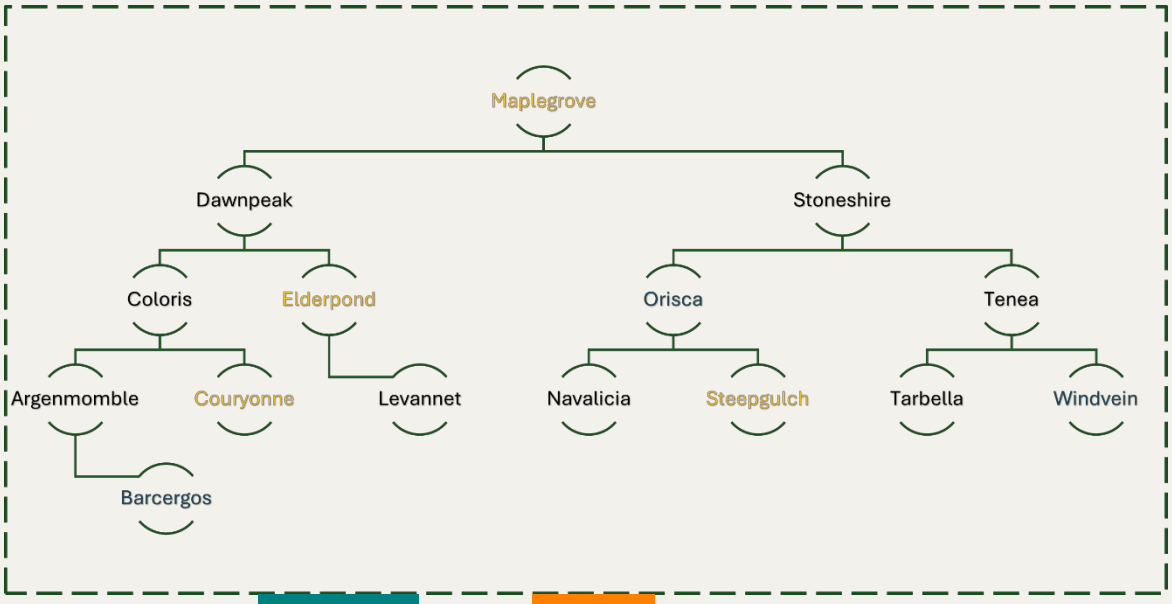
Her üçüncü Ada (Post-Order Traversal'a göre) ITEM::GOLDIUM ile doldurulur.

Ön Sipariş Geçişi

Her beşinci Ada (Ön Sipariş Geçişine göre) ITEM::EINSTEINIUM ile doldurulur.

BFS Ürün Düşüşü

AVL ağacının her 3. yeniden dengelenmesi, nadir ve en değerli ITEM::AMAZONITE'in mevcut eşyası olmayan ilk Ada'ya düşmesine neden olur. Bu, daha değerli eşyaların alt derinliklerde kalmasına yardımcı olur.



Şekil 5: EINSTEINIUM'dan önce GOLDIUM ile zenginleştirme sonrası ağaç.

2.3 Daha fazlası daha azdır: Aşırı Kalabalık

Diyar Şekillendiricileri çok güçlü oldukları için, çok fazla sayıda Şekillendiricinin bir yerde toplanması felaketlere neden olur. Her Adanın kaldırabileceği maksimum Şekillendirici kapasitesi vardır, bu sayıdan fazla Şekillendirici Adaya erişirse, Ada maalesef çöker ve kendini imha eder. Yıkılan Adalar Haritadan kaldırılmalı ve gerekirse AVL kurallarına uyarak dünya kendini yeniden dengelemelidir.

2.4 Dosyaya Kaydetme: Adalar ve Harita

Dünya Ağacı (Harita) iki farklı çıktı dosyasına kaydedilir: current_isles.txt ve current_map.txt. Bununla ilgili daha fazla bilgiyi Bölüm 4'te bulabilirsiniz.

3 Oyun Mekaniği: 5'e 1

3.1 Adım Adım Oynanış

Oyun simülasyonu aşağıdaki kayıtlar tarafından yönlendirilir:

1. **Erişim Günlükleri (access.log)**: Adalar'ı ziyaret etmeye veya işlemeye çalışan oyuncuların kayıtları.
2. **Düello Günlükleri (duels.log)**: Oyuncular tarafından başlatılan düelloların ve

sonuçlarının kayıtları. Oynanış aşağıdaki gibi ilerler:

- Access.log'daki her kayıt için:
  - Ada mevcutsa ve oyuncunun erişimi varsa, adayı ziyaret eder ve mevcut öğeyi toplar.
  - Ada mevcut değilse ancak oyuncunun yeterli Enerji Puanı varsa, yeni bir Ada oluşturulur ve Dünya Ağacına eklenir.
- Duels.log'daki her kayıt için:
  - Düello belirtilen sonuçla gerçekleşir ve Şekillendirici Ağacı buna göre güncellenir.
- Her beş erişim kaydı için bir düello gerçekleşir. Tüm erişim kayıtları işlendikten sonra hala düello kayıtları varsa, bu düellolar birbiri ardına yapılmalıdır.

Her eylemden sonra, güncellenmiş Şekillendirici Ağacı ve Dünya Ağacı terminalde görüntülenir.

3.2 Eğlenceli Ekstra: Tek Oyunculu Demo

Ödevin bu kısmı ekstra bir şey uygulamanızı gerektirmez

Ödeve eğlenceli bir ek olarak, oyun dünyası ve sıralamalarla doğrudan terminalden etkileşime girmenizi sağlayacak **tek oyunculu demo moduna** sahip olacaksınız**.** Bu modda oyuna kendi oyuncunuzu ekleyebilecek ve soruları yanıtlayarak düellolar kazanabileceksiniz. Sorular genel kültür ("Hacettepe Üniversitesi hangi yılda kurulmuştur?"), ağaçlar ("Dizi tabanlı ikili ağaçta sol çocuğun indeksi nedir?") ve mizah ("Python programcıları neden gözlük takar?") gibi çeşitli konulardan oluşmaktadır. Bunu sağlamak için size Questions.h, Questions.cpp ve Questions I/O dosyaları verilecektir. Bu dosyalardaki tüm fonksiyonlar çalıştırmanız için tam olarak uygulanmıştır.

Tek Oyunculu Demonun Özellikleri:

1. **İnteraktif Menü:** Oyuncunun gerçek zamanlı olarak düello yapma, keşfetme ve yeni Adalar oluşturma gibi eylemleri gerçekleştirmesini sağlayan menü odaklı bir sistem.
2. **Dinamik Güncellemeler**: Her eylemden sonra, program ASCII tabanlı bir ağaç görselleştirmesi kullanarak terminalde güncellenmiş Şekillendirici Ağacı ve Dünya Ağacını görüntüler.
3. **Düello**: Oyuncu kendi karakterini seçebilir ve diğerlerine meydan okuyabilir. Düelloların sonucu önceden belirlenmek yerine sorularla belirlenir.
4. **Keşif ve Üretim**: Oyuncular mevcut Adaları keşfetmeye çalışabilirler. Eğer erişimleri yoksa, puanlarını kullanarak yeni bir Ada oluşturabilirler.
5. **İlerleme Takibi**: Onur Puanları ve toplanan eşyalar her turdan sonra görüntülenir.

4 Dosyalar: Giriş/Çıkış

Aşağıda size verilen veya sizden istenen dosyalardan ayrıntılar ve örnekler bulabilirsiniz.

4.1 İlk Oyuncular: initial_realm_shapers.txt

Realm Shaper sınıfının beta testini yapacak oyuncular bu dosyada size verilmiştir. Oyuncular dosyada sıralanmıştır ve Şekillendirici Ağacına dosya ile aynı sırada eklenmelidir. Dosyada Oyuncu adı ve kayıtlı onur puanları bulunmaktadır.

Format:

```
#OyuncuAdı[tab]OnurPuanları
Oyuncu1 1800
Oyuncu2 1400
Oyuncu3 1450
Oyuncu4 1450
```

4.2 İlk Adalar: initial_world.txt

Dünyada var olan adalar bu dosyada size verilmiştir. Dosyadaki ada isimleri sıralı değildir, ancak dosyayı takiben haritaya tek tek eklenmelidir. Dosya sadece ada isimlerini içermektedir.

Format:

```
#IsleName
Maplegrove
Dawnpeak
Stoneshire
```

4.3 Erişim Kayıtları: access.log

Bu dosya daha önce oynanmış bir oyunun kayıtlarını içerir, bu kayıtları o oyunu yeniden oluşturmak ve uygulamanızı test etmek için kullanacaksınız. Dosya oyuncu adını ve oyuncunun değerlendirmeye çalıştığı adanın adını içerir. Ada zaten mevcutsa ve oyuncunun erişim hakları varsa, adayı ziyaret etmeli ve orada bulunan herhangi bir öğeyi toplamalıdır. Eğer ada mevcut değilse ve oyuncunun yeterli Şekillendirme Enerjisi varsa, belirtilen isimde yeni bir ada oluşturulmalı ve haritaya eklenmelidir.

Format:

```
#OyuncuAdı[tab]AdaAdı
#(eğer yer mevcutsa oyuncu ziyaret eder, değilse yeni yer eklenir (eğer oyuncunun erişimi varsa))
Oyuncu1 Couryonne
Oyuncu2 Coloris
Oyuncu3 Orisca
```

4.4 Düello Kayıtları: duels.log

Bu dosya daha önce oynanmış bir oyunun kayıtlarını içerir, bu kayıtları o oyunu yeniden oluşturmak ve uygulamanızı test etmek için kullanacaksınız. Dosya düelloyu başlatan oyuncunun adını ve düellonun sonucunu içerir.

Format:

```
#ChallangerPlayerName sonuç(1: kazandı 0:
kaybetti) Oyuncu101
Oyuncu4     0
```

4.5 Güncel Adalar: current_isles.txt

Bu, sağlamanız gereken bir çıktı dosyasıdır ve haritada hala var olan tüm adaları sırayla içermelidir. Bunlar isimlerine göre sıralanacaktır.

Format:

```
Argenmomble
Barcergos
Coloris
```

4.6 Güncel Harita: current_map.txt

Bu, mevcut haritanın yapısını içermesi gereken, sağlamanız gereken bir çıktı dosyasıdır. Haritanın düğümleri seviye seviye yazılacaktır. Aynı seviyedeki düğümler arasında bir [boşluk] olacaktır.

Format:

```
Maplegrove
Dawnpeak Stoneshire
Coloris Elderpond Orisca Tenea
Argenmomble Couryonne NULL Levannet Navalicia Steepgulch Tarbella Windvein
NULL Barcergos NULL NULL NULL NULL NULL NULL NULL NULL NULL NULL NULL NULL NULL
```

4.7 Mevcut Oyuncular: current_realm_shapers.txt

Bu çıktı dosyası, oyunda hala var olan tüm Realm Shapers'ın listesini içermelidir. Dosyada, oyuncuların isimleri sıralamalarına göre yazılmalıdır.

Biçim (Oyuncu4, ebeveynine (Oyuncu2) karşı bir düello kazandı):

```
Oyuncu1
Oyuncu4
Oyuncu3
Oyuncu2
```

4.8 Mevcut Oyuncular: current_shaper_tree.txt

Bu çıktı dosyası mevcut Şekillendirici Ağacının yapısını sağlamalıdır. Dosyada, oyuncuların adları ön sıra geçişine dayalı bir liste olarak yazılmalıdır.

Format (1-11 arası oyuncular birbiri ardına eklenir):

```
Oyuncu1
Oyuncu2
Oyuncu4
Oyuncu8
```

5 Atama Uygulama Görevleri ve Gereksinimleri: Kod

Bu bölümde, uygulamanız gereken bazı sınıflar ve işlevler özetlenmiştir. Başlangıç kodu şablon dosyalarında verilen işlevlerin adlarını veya imzalarını değiştirmeyin. Aynı şekilde, belirtilen üye değişkenlerin türlerini değiştirmekten veya yeniden adlandırmaktan kaçının. Bunun dışında, ek işlevler sunmanız gerekecek ve gerektiğinde değişken değişkenler ekleyebilirsiniz.

5.1 `RealmShaper Sınıfı`

Bu sınıf, yeni Adalar yaratabilen özel bir oyuncu sınıfı olan Diyar Şekillendiricisini temsil eder.

- **Nitelikler**:
  - string name: Oyuncunun adı.
  - int honourPoints: Oyuncunun mevcut Onur Puanı, işçilik ve düello için elverişliliğini belirler.
  - int toplananEnerjiPuanları: Öğelerden elde edebildikleri puanlar.
- **İnşaatçılar:**
  - RealmShaper(const string& playerName, int initialHonour) Belirli bir isim ve başlangıç Onur Puanı ile bir Realm Shaper başlatır.
  - RealmShaper() Varsayılan niteliklere sahip boş bir Realm Shaper'ı başlatan varsayılan kurucu.
- **Kopya Oluşturucu**: RealmShaper(const RealmShaper& other) Diğer Realm Shaper nesnesinin bir kopyasını oluşturur.
- **Operatör Aşırı Yüklemesi**:
  - '==' operatörü iki oyuncuyu isimlerine göre karşılaştırır.
  - '<<' operatörü, Realm Shaper'ların isimlerinin çıktısını almaya yardımcı olur.
- **Fonksiyonlar**:
  - void onur kazan(int puan)
    Oyuncunun Onur Puanını belirtilen miktarda artırır.
  - voidOnurKaybet(int puan)
    Oyuncunun Onur Puanını belirtilen miktarda azaltır. Onur Puanı sıfıra ulaşırsa, oyuncu oyundan çıkarılır.
  - void hasEnoughEnergy()
    Oyuncunun yeni bir Isle yapmak için yeterli enerjisi olup olmadığını kontrol eder.
  - void collectItem(Item item)
    Eşyayı toplar ve değerini oyuncunun enerji puanlarına ekler.
  - static vector<RealmShaper *> readFromFile(const std::string &filename)
    Oyuncu bilgilerini giriş dosyasından okur.

5.2 `Ada Sınıfı`

Bu sınıf, dinamik oyun dünyasını oluşturan AVL ağacındaki bir düğüm olan bir Ada'yı temsil eder.

- **Nitelikler**:
  - dize adı

    Adanın adı.
  - ITEM öğesi

    Ada üzerinde bulunan öğe (örneğin, GOLDIUM, EINSTEINIUM, AMAZONITE).
  - int kapasite

    Bir seferde Ada'yı ziyaret edebilecek maksimum Diyar Şekillendirici sayısı.
  - int playerCount

    Ada'daki mevcut Diyar Şekillendirici sayısı.
- **İnşaatçılar:**
  - Isle(const string& isleName)

    Verilen ada sahip ve ilişkili öğesi olmayan bir Isle oluşturur.
  - Isle(const string& isleName, ITEM itemType)

    Verilen isimde bir Isle oluşturur ve ona belirli bir öğe atar.
- **Kopya Oluşturucu**:

  Isle(const Isle& other)

  Başka bir Isle nesnesinin kopyasını oluşturur.
- **Operatör Aşırı Yüklemesi**:
  - '==' işleci iki Ada'yı adlarına göre karşılaştırır.
  - '<' ve '>' operatörleri AVL ağacında sıralama için iki Ada'yı karşılaştırır (isme göre alfabetik olarak).
  - '<<' operatörü Adaların isimlerine göre çıktı almasına yardımcı olur.
- **Fonksiyonlar**:
  - void assignItem(ITEM newItem)

    Ada'ya yeni bir öğe atar.
  - static vector<Isle*> readFromFile(const std::string &filename)

    Giriş dosyasından Isles'ı okur ve ayrıştırır.
  - bool increasePlayerCount()

    Kapasite dolu değilse oyuncu sayısını artırır. isFull değerini döndürür.

5.3 `Harita Sınıfı`

Bu sınıf, oyun dünyası haritasını yönetmek için kullanılan AVL Ağacı yapısını temsil eder.

5.3.1 MapNode Yapısı

Bu yapı, bir Isle nesnesi ve alt düğümlere işaretçiler içeren harita yapısındaki bir düğümü temsil eder.

- Nitelikler:

* Ada* ada: Düğümde depolanan veriler.
* MapNode* left: Sol çocuğa işaretçi.
* MapNode* right: Sağ çocuğa işaretçi.
* int yükseklik: Düğümün yüksekliği.

– **İnşaatçılar:**

MapNode(Isle* isle) Bir MapNode'u verilen Isle nesnesi ile başlatır.

– **Yok edici**:

~MapNode() Sahiplik burada yönetiliyorsa, Isle nesnesi için dinamik olarak ayrılmış belleği temizler.

5.3.2 Harita Sınıfı

- **Nitelikler**:
  – Isle* root: AVL ağacının kök düğümüne işaretçi.
- **Yapıcılar**: Map() Boş bir AVL ağacını başlatan varsayılan kurucu.
- **Fonksiyonlar**:
  – void insert(Isle *isle)
  Ağaca yeni bir Isle ekler. Gerekirse ağacı dengeler.
  – void remove(Isle *isle)
  Bir Ada'yı ağaçtan kaldırır, daha sonra yeniden dengeler.
  – Ada* findIsle(const string& isleName)
  İsmine göre bir Ada arar ve bulunursa Ada'ya bir işaretçi döndürür.
  – MapNode* findNode(Isle isle)
  Bir düğümü tuttuğu Isle'a göre arar ve bulunursa düğüme bir işaretçi döndürür.
  – int getNodeDepth(Isle *isle)
  Ağaç içindeki bir Isle'ın derinlik seviyesini belirler.
  – int getDepth()
  Tam ağacın derinlik seviyesini belirler.

  - void inOrderTraversal()
    Ağacı alfabetik sırayla dolaşır.
  - void preOrderTraversal()
    Ağacı Geziyor ön siparişte.
  - void postOrderTraversal()
    Ağacı son sırada dolaşır.
  - void populateWithItems()
    Tanımlanan kurallara göre dünyayı öğelerle zenginleştirir.
  - void writeToFile(const std::string &filename)
    Ağaç yapısını Bölüm 4.6'da belirtildiği gibi bir dosyaya yazın.
  - void writeIslesToFile(const std::string &filename)
    Adaları Bölüm 4.5'te belirtildiği gibi bir dosyaya yazın.

- **Yıkıcı**: ~Map() AVL ağacındaki tüm düğümleri temizler.

5.4 `ShaperTree Sınıfı`

Bu sınıf, oyuncuları sıralamak için kullanılan ağacı temsil eder.

Ö**znitelikler**: vector<RealmShaper*> players: Realm Shaper nesnelerine işaretçilerden oluşan dinamik bir dizi.

Yapıcılar: ShaperTree() Boş bir oyuncu sıralama ağacını başlatan varsayılan kurucu.

Fonksiyonlar:

- void insert(RealmShaper* newPlayer)
  Bir sonraki uygun pozisyonda ağaca yeni bir oyuncu ekler.
- void remove(RealmShaper* player)
  Verilen sıradaki oyuncuyu kaldırır ve ağacın bütünlüğünü korur.
- void düello(RealmShaper *challenger, bool sonuç)
  İki oyuncu arasında bir düelloyu simüle eder ve meydan okuyan kazanırsa pozisyonlarını değiştirir.
- int getDepth(RealmShaper* player)
  Bir oyuncunun ağaç içindeki derinlik seviyesini belirler.
- int getDepth()
  Tam ağacın derinlik seviyesini belirler.
- RealmShaper* findPlayer(const string& playerName)
  Bir oyuncuyu ismine göre arar ve bulunursa oyuncuya bir işaretçi döndürür.
- int findPosition(RealmShaper* player)
  Ağaç dizisinde bir oyuncu arar, bulunursa konum indeksini döndürür.
- void inOrderTraversal()
  Ağacı sırayla dolaşır.

- void preOrderTraversal()
  Ağacı Geziyor ön siparişte.
- void postOrderTraversal()
  Ağacı son sırada dolaşır.
- void levelOrderTraversal()
  Ağacı kademe kademe gezer.
- void writeToFile(const std::string &filename)
  Şekillendirici ağaç oyuncularını bir dosyaya yazar, bkz. Bölüm 4.8.
- void writePlayersToFile(const std::string &filename)
  Oyuncu sıralamalarını bir dosyaya liste olarak yazın, Bölüm 4.7'ye bakın.

Yıkıcı: ~ShaperTree() Dinamik olarak ayrılmış Realm Shaper nesnelerini temizler ve vektörü temizler.

5.5 `Oyun Dünyası Sınıfı`

Bu sınıf kapsayıcı oyun dünyasını temsil eder, hem Şekillendirici Ağacı hem de Dünya Ağacını yönetir ve bunlar arasındaki etkileşimleri kolaylaştırır.

Nitelikler:

- Harita worldTree: Dinamik oyun dünyası haritasını temsil eden AVL ağacı.
- ShaperTree shaperTree: Oyuncu sıralama sistemini temsil eden ağaç.

Kurucular: GameWorld() Varsayılan ağaçlarla boş bir oyun dünyası başlatır.

Fonksiyonlar:

- void initializeGame(vector<Isle*> isles, vector<RealmShaper*> players) Şekillendirici Ağacı ve Dünya Ağacını sağlanan Adalar ve oyuncularla doldurarak oyun dünyasını başlatır.
- void processGameEvents(const string &accessLogs, const string &duelLogs) Erişim ve düello günlüklerindeki eylemleri uygulayarak oyunu işler ve her iki ağacı dinamik olarak günceller.
- void displayGameState()
  ASCII tabanlı görselleştirme kullanarak terminaldeki Dünya Ağacı ve Şekillendirici Ağacının mevcut durumunu görüntüler.
- void saveGameState(args*);
  Dünya Ağacı ve Şekillendirici Ağacının mevcut durumunu çıktı dosyalarına kaydeder.
- bool hasAccess(RealmShaper* player, Isle* isle)
  Verilen oyuncunun rütbesine göre belirtilen Isle'a erişimi olup olmadığını belirler.
- void exploreArea(RealmShaper* playerID, Isle* isle)
  Oyuncunun belirtilen bir adayı keşfetmesini sağlar. Ada mevcutsa, oyuncu onunla etkileşime girer. Eğer yoksa, oyuncunun puanlarına bağlı olarak yeni bir Ada oluşturulabilir.

Kullanılması Gereken Başlangıç Kodu

Bu başlangıç kodunu kullanmalısınız. Tüm sınıflar doğrudan **zip** arşivinizin içine yerleştirilmelidir.

Yapılandırma Oluşturma ve Çalıştırma

İşte kodunuzun nasıl derleneceğine dair bir örnek (main.cpp yerine test dosyalarımızı kullanacağımıza dikkat edin):

```
$ g++ -std=c++11      *.cpp, *.h -o HUSLAND
```

Veya kodunuzu derlemek için örnek girdi içinde sağlanan Makefile veya CMakeLists.txt dosyasını kullanabilirsiniz:

```
$ make
```

veya

```
$ mkdir HUSLAND_build
$ cmake -S . -B HUSLAND_build/
$ make -C HUSLAND_build/
```

Derleme işleminden sonra programı aşağıdaki şekilde çalıştırabilirsiniz:

```
$ ./HUSLAND initial_world.txt initial_realm_shapers.txt access.log duels.log
↪  current_isles.txt current_map.txt current_realm_shapers.txt
↪  current_shaper_tree.txt
```

Notlandırma Politikası

- **Bellek sızıntısı ve hata yok: 10%**
  - Bellek sızıntısı yok: 5%
  - Bellek hatası yok: 5%
- **Diyar Şekillendiricileri Ağacının doğru uygulanması: %25**
  - Giriş dosyalarından uygun ağaç başlatma %5
  - Düelloların uygulanması %5
  - Oyuncuların çıkarılmasının uygulanması %5
  - Erişim hesaplamasının uygulanması %5
  - Çapraz geçişlerin uygulanması %5
- **Dünya Ağacı'nın (Harita) doğru uygulanması: 45%**
  - Giriş dosyalarından uygun ağaç başlatma %5
  - Oyuncu 5%'ten Ada ekleme uygulaması
  - Aşırı kalabalıklaşma sonrası ada silme uygulaması %10
  - Ürün dağıtımının uygulanması %10
  - Isle erişiminin uygulanması %10
  - Ağacı sırayla dosyaya kaydetme uygulaması %5
- **Oyun mekaniğinin doğru uygulanması: 10%**
  - Günlük dosyalarından düzgün oyun başlatma %5
  - Oyunun uygulanması %5
- **Çıkış testleri: 10%**

Gönderimin kabul edilmesi için ödevden SIFIR OLMAYAN bir not almanız gerektiğini unutmayın. Notu 0 olan gönderimler GÖNDERİM YOK olarak sayılacaktır!

Önemli Notlar

- Son başvuru tarihini kaçırmayın: **15/12/2024 Pazar (23:59:59)** .
- Ödeve not verilene kadar tüm çalışmalarınızı saklayın.
- Gönderdiğiniz ödev çözümü sizin özgün, bireysel çalışmanız olmalıdır. Kopya veya benzer ödevlerin her ikisi de kopya olarak değerlendirilecektir.
- Sorularınızı Piazza aracılığıyla sorabilirsiniz (https://piazza.com/hacettepe.edu.tr/fall2024/bbm203) ve Piazza'da tartışılan her şeyden haberdar olmanız gerekiyor.
- Kodunuzu **Tur^3 Bo Grader** https://test-grader.cs.hacettepe.edu. tr/ üzerinden test **etmelisiniz** (**gönderim olarak sayılmaz!**).
- Çalışmanızı https://submit.cs.hacettepe.edu.tr/ adresi üzerinden dosya ile birlikte göndermeniz gerekmektedir.

  hiyerarşisi aşağıda verilmiştir:
  - **b<studentID>.zip**
    - GameWorld.cpp *<DOSYA>*
    - GameWorld.h *<DOSYA>*
    - Isle.cpp *<DOSYA>*
    - Isle.h *<DOSYA>*
    - Map.cpp *<DOSYA>*
    - Map.h *<DOSYA>*
    - RealmShaper.cpp *<DOSYA>*
    - RealmShaper.h *<DOSYA>*
    - RealmShapers.cpp *<DOSYA>*
    - RealmShapers.h *<DOSYA>*
- **Bu başlangıç kodunu kullanmalısınız**. Tüm sınıflar doğrudan **zip** arşivinize yerleştirilmelidir.
- Bu dosya hiyerarşisi gönderilmeden önce sıkıştırılmalıdır (.rar değil, yalnızca .zip dosyaları desteklenir).
- Herhangi bir nesne veya çalıştırılabilir dosya göndermeyin. Yalnızca başlık ve C++ dosyalarına izin verilir.

Akademik Dürüstlük Politikası

Ödevler üzerindeki tüm çalışmalar **bireysel olarak yapılmalıdır**. Verilen ödevleri sınıf arkadaşlarınızla tartışmanız teşvik edilmektedir, ancak bu tartışmalar soyut bir şekilde yürütülmelidir. Yani, belirli bir problemin belirli bir çözümüyle ilgili tartışmalara (gerçek kodda veya sözde kodda) **tolerans gösterilmeyecektir**. Kısacası, bir başkasının çalışmasını (internette bulunan çalışmalar **da** dahil olmak üzere) tamamen veya kısmen kendi çalışmanız olarak teslim **etmeniz akademik dürüstlüğün ihlali** olarak değerlendirilecektir. Web'de bulunan her şey bir başkası tarafından yazıldığından, önceki koşulun web'de bulunan materyal için de geçerli olduğunu lütfen unutmayın.

Başvurular benzerlik kontrolüne tabi tutulacaktır. Benzerlik kontrolünden geçemeyen başvurulara not verilmeyecek ve akademik dürüstlük ihlali vakası olarak etik kurula bildirilecek, bu da ilgili öğrencilerin uzaklaştırılmasıyla sonuçlanabilecektir.