Eindopdracht C++2

Maak in C++ een applicatie van het spel *Machiavelli*. Zie de bij deze eindopdracht geplaatste pdf voor de spelregels*. Je moet **in tweetallen**** een console applicatie maken die door middel van tekst weergeeft wat jouw gelegde kaarten zijn en/of wat de gelegde kaarten van de tegenspeler zijn. Het is niet verboden om ergens een grafische weergave van te hebben, maar je krijgt hier geen extra punten voor. De console geeft ook aan wat voor keuzes een speler heeft indien deze een keuze moet maken.

Voorbeeld:

*Machiavelli is normaalgesproken een spel voor 2 tot 7 spelers, maar voor deze opdracht beperken we ons tot de variant met 2 spelers.

** **Er moet wederom in duo's gewerkt worden. Alleen na nadrukkelijke toestemming van je practicum docent mag je in je eentje de eindopdracht doen. Het is niet toegestaan de eindopdracht met drie of meer mensen te doen.**

Gegeven

- Een scan van alle verschillende kaarten is bij deze opdracht gegeven. Deze scans zijn alleen ter kennisneming, de afbeeldingen mogen verder niet gebruikt of verspreid worden (om auteursrechtelijke redenen).
- Voor deze opdracht is een tekstbestand gegeven waarin alle bouwkaarten staan met hun naam, hun waarde, hun kleur, en eventueel hun speciale eigenschap. Sommige kaarten komen meer dan één keer voor in het spel. Kaarten die in het echte spel meerdere keren voorkomen staan ook meerdere keren in dit bestand. Na inlezen heb je dus vanzelf de juiste aantallen van elke kaart.
- Voor deze opdracht is een file gegeven waarin de acht karakterkaarten staan met hun volgordenummer.

- Er is voorbeeldcode gegeven ter ondersteuning van de implementatie van bepaalde eisen. Deze voorbeeldcode mag vrijelijk gebruikt of aangepast worden in de opdracht.
- De koningskaart (de kaart van de kroon die normaalgesproken op een voetstukje staat) hoeft niet als kaart geïmplementeerd te worden. Het enige wat zichtbaar moet zijn is een regeltje tekst waarin aangegeven wordt of de speler in kwestie koning is.
- De overzichtskaart hoeft niet als kaart geïmplementeerd te worden. De tekst op de kaart hoeft alleen voor te komen als “hulptekst”. Wanneer er dus voor de hulpoptie gekozen wordt moet de tekst van de kaart weergegeven worden.

Minimum Eisen

De volgende eisen zijn **minimaal** verplicht om te implementeren. **Wanneer één of meer van deze eisen niet geïmplementeerd zijn krijg je automatisch een 1 voor de eindopdracht.**

1. Vóór het assessment begint moet de laatste versie van de opdracht zijn **ingeleverd op blackboard** voor beide studenten. (indien gewenst hoeft dat pas te gebeuren in de laatste minuut voor het assessment plaats vindt)
2. **Je programma mag niet crashen**. Gebeurt dat toch, dan ter plekke repareren of anders deze minimum eis niet voldaan.
3. **Je schrijft een game server**, die je geheel mag baseren op de voorbeeldcode. Je bent niet verplicht om een speciale client te bouwen; je kunt daarvoor gewoon telnet gebruiken. Zie Blackboard voor extra informatie over telnet.
4. Het moet **volledig in C++** geprogrammeerd zijn.
5. Je mag **alleen** van de **Standard Library** gebruik maken, plus eigen code. Het gebruik van externe libraries zoals Boost of Poco is niet toegestaan. Die heb je ook helemaal niet nodig: alles kan in principe gedaan worden met de mogelijkheden die de Standard Library je biedt. (Tip: maak goed en veel gebruik van cppreference.com).
6. Er mag ***alleen*** gebruik worden gemaakt van **smart pointers** (unique_ptr, shared_ptr, en weak_ptr). "Gewone" pointers mogen dus niet voorkomen!
   Let op: dit gaat natuurlijk alleen over pointers die naar heap-objecten wijzen. Als je stack-based of global objects hebt, hoeven die niet in smart pointers verpakt te worden. Doorgeven van het adres van een dergelijk object is dan weliswaar een ruwe pointer, maar niet eentje die met new is gemaakt en met delete moet worden opgeruimd. Die zijn gewoon toegestaan (hoewel je dan nog steeds goed moet opletten wat de life time en scope van deze objecten zijn, anders riskeer je dangling pointers…).
7. Er moet een vorm van *memory leak detection* beschikbaar zijn ten tijde van het assessment.
8. Er mogen geen memory leaks voorkomen (maar dat krijg je dus cadeau bij correct gebruik van smart pointers).
   Notabene: Memory leaks waar jullie niets aan kunnen doen mogen wel voorkomen, maar dan moeten jullie wel kunnen beredeneren dat deze niet veroorzaakt worden door jullie code. “False positives” (de memory leak detection tool geeft een leak aan, maar dit is niet correct) mogen ook voorkomen. In geval van twijfel vraag tijdig even de mening van je practicum docent.
9. Het programma moet netjes beëindigd kunnen worden door middel van bijvoorbeeld een *quit* commando. (De console weg kunnen klikken met het kruisje volstaat dus niet.)
10. Alle code moet *exception safe* zijn. Dat wil zeggen dat het niet mogelijk moet zijn dat memory leaks ontstaan als gevolg van een exception. (En ook dit krijg je cadeau wanneer je de smart pointers goed toepast.)
11. Er moet **exception handling** plaatsvinden. De applicatie mag *nooit* een exception aan de gebruiker doorgeven, dit moet door middel van een berichtje in de console weergegeven worden, waarna, mocht dit nodig zijn, netjes uit het programma gesprongen wordt. (In het

voorbeeldprogramma is de exception handling op het hoogste niveau al helemaal geregeld, dus als je hierop voortborduurt zul je nooit zomaar op een exception crashen.)

12. Er moet van minstens één **STL container**, van minstens één **STL algoritme**, en van **STL iterators** gebruik worden gemaakt. Je algoritme moet door middel van een **lambda expression** worden aangeroepen.
13. De bouwkaarten en de karakterkaarten moeten beide door middel van een **file stream** worden ingelezen. Hierbij hoort ook een correcte implementatie van input en output operators. Dus je moet cin >> myObject; en cout << myObject << endl; ondersteunen.
    (Als je niet gebruik wilt maken van de gegeven bestanden mag je het ook op een andere manier oplossen. We willen echter wel graag zien dat je tekstbestanden van enige complexiteit correct kunt verwerken.)
14. Als je casts gebruikt mag er ***alleen*** gebruik worden gemaakt van **C++ style casts** (static_cast, const_cast, dynamic_cast, en reinterpret_cast), of de varianten hiervoor die shared_ptr je biedt.

Bonuspunten

1. **De paarse kaarten** zijn lastig te implementeren, omdat ze qua spelregels op allerlei plekken in je software architectuur ingrijpen. De moeilijkste kaart is in dit verband: het **Kerkhof**, en die levert je +0,5 punt op als je hem correct hebt ingebouwd. Als je ook de rest van de paarse kaarten goed hebt geïmplementeerd verdien je nog eens +0,5 punt; in totaal dus +1 punt extra te verdienen met het volledig implementeren van alle paarse kaarten.
2. **De toestand van het spel wordt constant opgeslagen** d.m.v. file streaming. Wanneer het spel wordt onderbroken (of onverhoopt crasht) moet de toestand weer ingeladen kunnen worden en verder gegaan waar men was gebleven. Correcte implementatie hiervan levert +2 punten. Vergis je niet, dit is lastiger dan je denkt. Uitdaging!

Maluspunten

1. De code is niet volledig ***const correct***: -1 punt
2. Niet goed kunnen verantwoorden waarom je pointers gebruikt in plaats van objecten. In geval van twijfel vraag tijdig even de mening van je practicum docent.
   Bijvoorbeeld: zonder goede reden gebruik maken van Vector<unique_ptr<MyClass>> in plaats van Vector<MyClass> : -0,5 punt.

Beoordeling

Indien alle minimum eisen foutloos geïmplementeerd zijn wordt de eindopdracht met een 7 beoordeeld. Dit cijfer kan verder worden verhoogd en/of verlaagd door de zaken die genoemd staan bij Bonuspunten en Maluspunten.

Beoordelen van “nette code”

Zoals je uit bovenstaande punten al hebt kunnen afleiden vind er geen beoordeling plaats van je ontwerp of je architectuur en wordt er niet gekeken of je code wel “netjes” is. Dit is niet omdat dit niet belangrijk wordt geacht in C++ of door de docenten. In tegendeel, dit vinden we altijd een belangrijk punt. Echter is dit niet wat er getoetst wordt bij het vak C++ 2 (Dit wordt meer getoetst door bijvoorbeeld het vak Advanced Design Patterns). Dat wil natuurlijk niet zeggen dat je geen baat hebt bij het netjes programmeren van je eindopdracht! Integendeel, je zult daar alleen maar plezier van ondervinden omdat er stabielere, overzichtelijkere code wordt geproduceerd die zich bijvoorbeeld beter leent om problemen te vinden en op te lossen.

Belangrijke tip: cheat-mode inbouwen om setup over te kunnen slaan

Het spelverloop zoals vastgelegd in de spelregels maakt dat je steeds moet beginnen met een setup-ronde, waar spelers hun karakters kiezen. Pas als dit klaar is begint de speelronde pas echt. Dit is vervelend met testen, als je even snel wilt controleren of een stukje functionaliteit werkt.

Tip: bouw een cheat-mode in, waarbij je met een speciaal commando direct in de spelronde terecht komt, en er al een verdeling van karakters heeft plaatsgevonden (al dan niet random).