Atividade 3: A* é uma modificação de um algoritmo de busca. Qual o algoritmo usado como base? Qual a modificação proposta por A*? Podemos dizer que A* é uma abordagem gulosa? Em que tipo de jogo ele deve ser usado? Dê exemplos de caso ótimo e péssimo para o A*.

Resposta: O A* usa como base o algoritmo de dijkstra, contendo uma modificação a qual seria de o A * tenta procurar um caminho melhor usando uma função heurística que prioriza os nós que deveriam ser melhores que os outros. O que separa o A* de uma abordagem gulosa seria que o mesmo leva em consideração o custo/distância já percorrida.
Tipo de Jogo.
Exemplo: Tower defense (TD) é um gênero de videogames de estratégia onde o objetivo de jogos de tower defense é tentar impedir que os inimigos percorram por um mapa, por meio de armadilhas para atrasá-los e torres que atiram neles enquanto passam.
Assim o A* que geralmente é usado para encontrar o menor caminho entre dois pontos pode ser usado para cada inimigo para encontrar um caminho que o leve a alcançar esse objetivo.

Atividade 4: Um problema clássico em jogos está em suas políticas de se dar recompensa. Resumindo a missa: recompensas não devem ser puramente randômicas, por um lado jogadores tem que ter a chance de ganhar bons prêmios a qualquer momento, por outro lado eles não podem ganhar o tempo todo nem passar muito tempo sem ganhar. Quais modificações você implementaria ao random para melhorar essa experiência?

Resposta:
Modificações:
Definiria um intervalo de tempo para controlar o tempo em que o random dá recompensas.
Dividiria as recompensas em níveis de qualidade(Excelente, boa, média, ruim … sem recompensa), as quais seriam dadas pelo random dependendo das jogadas do player. A recompensa seria determinada randomicamente entre uma faixa de níveis de qualidade.
Ex: Jogadas do Player X determinam que ele ganhe uma recompensa que não seja ruim. O random pegaria uma recompensa entre Recompensas Excelentes, Boas e médias com uma porcentagem para cada faixa (Excelente 5%, boa 15% média 80%).
Depois de receber a recompensa o intervalo de tempo a ser esperado para outra recompensa seria determinado pela qualidade da recompensa dada. Quanto melhor a recompensa mais tempo será esperado pelo random.

Atividade 11: Em um cenário hipotético uma operação de jogos tem várias e várias GB de logs salvos em arquivos de texto(em formato semi-estruturado). Devido um problema no servidor, alguns jogadores receberam o dobro de itens de uma determinada mecânica do jogo. Quais passos vc faria para detectar esses jogadores?

Resposta:

Algoritmo Detectar_Falha_Dobro_Itens:
Inicialização:
Carregar os dados dos arquivos em texto de forma que podemos lê-los.
para cada linha dentro do texto:
adicionar a linha na lista Linhas
Sabendo que a 1 linha (1 objeto da lista é a lista contendo os headers de informações tais como usuario, itensX, itensY....)
para cada linha em Linhas:
verificar se é a primeira, Linhas[0]:
se sim faça: split na linha e adicione o resultado na lista Chaves.
se não faça: split na linha e adicione o resultado na lista Valores.
Crie uma lista de dicionários chamado Dados usando os valores das listas Chaves e valores.
Percorra Dados:
Salve numa lista chamada itensMecanicaX o nome do usuário e o campo contendo a quantidade de itens daquela mecânica itensX de cada dicionário dentro da lista UserItensX, como uma tupla (usuário,itensX).
para cada Tupla em UserItensX:
Verifique quais possuem o campo itensX com valor dobrado
Guarde os nomes dos usarios na lista Nomes
retorne Nomes

Atividade 12: Em um parágrafo descreva as vantagens e desvantagens de CI(Continuous Integration) na operação de software

Resposta:
Na integração contínua tentamos estar sempre com a versão final do projeto atualizada e sem erros, para que isso ocorra a equipe de desenvolvedores precisará escrever testes automatizados para cada novo recurso, melhoria ou correção de bug e os desenvolvedores também precisam mesclar suas alterações do projeto no ramo principal o mais rápido possível além de manter um servidor de integração contínua que possa ficar monitorando o repositório principal executando os testes automaticamente para cada alteração feita. Feito tudo isso, a integração contínua nos garante que menos bugs serão enviados para produção visto que os mesmos foram capturados antecipadamente pelos testes e corrigidos pelos desenvolvedores que ao notarem que suas alterações foram quebradas podem trabalhar para corrigi-las antes de irem para outra tarefa, além de que, com os testes automatizados e executando no servidor, a equipe de controle fica livre para se concentrar na qualidade. Assim a criação da versão final é mais fácil, visto que os problemas de integração foram corrigidos anteriormente.