Blockchains και Smart Contracts
Περιγραφή εργασίας εξαμήνου

Φίλιππος Δουραχαλής, f3312205

CryptoSOS:

Η υλοποίηση της απλής εκδοχής του παιχνιδιού ήταν σχετικά απλή, όσον αφορά την λογική του smart contract. Η μεγαλύτερη δυσκολία που αντιμετωπίστηκε σε αυτό το σημείο ήταν η απόφαση για το πως θα αποθηκεύει και θα χειρίζεται το πρόγραμμα την κατάσταση του παιχνιδιού. Η ιδέα αρχικά ήταν να χρησιμοποιηθεί απευθείας ένα String για την αποθήκευση της κατάστασης, προκειμένου να γίνεται εύκολα η επεξεργασία της και ο έλεγχός της σε κάθε move ή getGameState, όπως θα μπορούσε να γίνει σε μια γλώσσα όπως η Java με την μέθοδο charAt(int index) των Strings και η forEach για την προσπέλαση των χαρακτήρων τους.

Σύντομα όπως προέκυψε το σημαντικό πρόβλημα του τρόπου με τον οποίο χειρίζεται η solidity τα Strings και η εμφανής έλλειψη μεθόδων για την επεξεργασία τους, λόγω του αυξημένου κόστους που έχουν τέτοιου είδους πράξεις. Για τον λόγο αυτό και για να ελαχιστοποιηθεί το κόστος σε gas των μεθόδων, επιλέχθηκε η λύση της αποθήκευσης της κατάστασης σε ένα fixed-length uint array, όπου κάθε αριθμός αντιπροσωπεύει ένα από τα δυνατά σύμβολα που μπορούμε να έχουμε (0 για το «-», 1 για το «S» και 2 για το «O»). Έτσι, όταν ένας χρήστης κάνει μια κίνηση, προσδιορίζοντας το αντίστοιχο κελί, αρκεί να ελέγξουμε αν η θέση του πίνακα που ανταποκρίνεται σε αυτό είναι 0 και αν ναι, να την αλλάξουμε, χωρίς να χρειάζονται περαιτέρω μετατροπές ή έλεγχοι.

Όταν ένας χρήστης ζητήσει να λάβει την κατάσταση ενός παιχνιδιού, τότε μόνον χρειάζεται να διατρέξουμε τον πίνακα με ένα for loop και να ανασυνθέσουμε ένα string βάσει ενός mapping από ακέραιους στους δυνατούς χαρακτήρες του παιχνιδιού.

MultiSOS:

Για την υλοποίηση των παράλληλων παιχνιδιών και εδώ η βασική ιδέα ήταν σχετικά απλή. Όλες οι μεταβλητές που χρειάζονται για την παρακολούθηση μιας συγκεκριμένης παρτίδας (διευθύνσεις παικτών, χρονόμετρο, πίνακας συμβόλων), τοποθετούνται σε ένα struct, το οποίο γίνεται map σε ένα συγκεκριμένο gameId μέσω ενός mapping. Όλες οι μέθοδοι που χρειάζεται να έχουν πρόσβαση στο state ενός συγκεκριμένου παιχνιδιού (π.χ. οι move, getGameState, cancel κτλ.) λαμβάνουν το gameId ως παράμετρο ώστε να γνωρίζουν ποιο παιχνίδι να λάβουν από το mapping. Το gameId προφανώς γνωστοποιείται και στους παίκτες μέσω των events όταν ξεκινάει μια νέα παρτίδα μεταξύ τους, ώστε να μπορούν να το δώσουν στις αντίστοιχες μεθόδους.

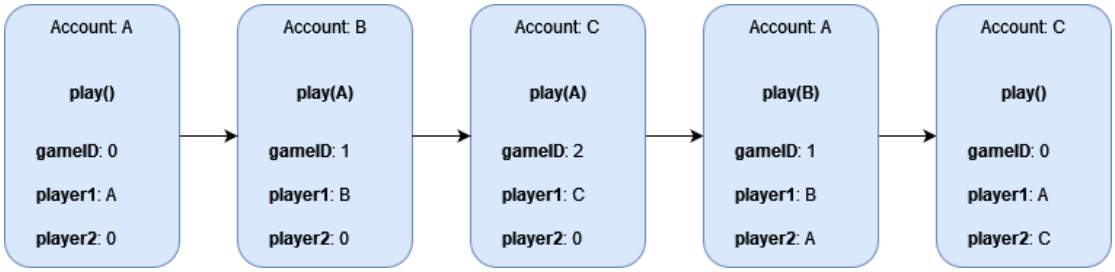
Για τα private lobby η πρώτη δυσκολία ήταν το πως θα μπορέσει το contract να παρακολουθεί το ποιοι παίκτες θέλουν να παίξουν με ποιους. Η λογική της υλοποίησης εδώ στηρίζεται στα invitations. Τα invitations δεν είναι τίποτα άλλο παρά ένα mapping από address σε ένα δυναμικό array από uint (ώστε ένας παίκτης να μπορεί να λάβει προσκλήσεις από πολλούς -ενδεχομένως και τους ίδιους- παίκτες), τα οποία αναπαριστούν τα gameIds των παιχνιδιών. Κάθε παίκτης που καλεί την play(address rival) ελέγχει τον πίνακα με τις προσκλήσεις του και εξετάζει για κάθε κατάσταση παιχνιδιού που ανταποκρίνεται στα gameId που έχει, αν κάποια έχει ως player1 τον rival (δηλ. αν ο rival τον περιμένει ήδη σε ένα ιδιωτικό παιχνίδι). Αν ναι, αποδέχεται την πρόσκληση, θέτοντας τον εαυτό του ως player2 και το παιχνίδι ξεκινάει. Αλλιώς φτιάχνει με τη σειρά του μια πρόσκληση προς τον rival και περιμένει. Σημειώνεται ότι στην δεύτερη περίπτωση ο καθολικός μετρητής που παρακολουθεί τα gameIds (currentGameId) αυξάνεται αμέσως μετά την δημιουργία ενός νέου παιχνιδιού, ακόμα κι αν δεν έχει μπει ο player2 ακόμα. Αυτό γίνεται για δύο λόγους:

a) Ο Player2 γνωρίζει εκ των προτέρων πως να βρει την κατάσταση ενός παιχνιδιού στο οποίο έχει προσκληθεί, αφού το gameId του θα βρίσκεται στον πίνακα των προσκλήσεων. Ακόμα κι αν το gameId αυξηθεί (π.χ. λόγω δημιουργίας άλλων παράλληλων παιχνιδιών στο ενδιάμεσο), μπορεί να εντοπίσει το συγκεκριμένο παιχνίδι στο οποίο περιμένει ο rival του μέσω του πίνακα. Αυτό μας φέρνει στον επόμενο (και πιο σημαντικό) λόγο.
b) Αν ο μετρητής σε ένα ιδιωτικό παιχνίδι δεν αυξηθεί αμέσως, αλλά μόνο όταν μπει και ο player2, ουσιαστικά τον **δεσμεύουμε** επ' αόριστον με αποτέλεσμα να μην μπορούν να δημιουργηθούν άλλα public ή private παιχνίδια μέχρι να ξεκινήσει το συγκεκριμένο. Αυξάνοντας τον αμέσως στα private (όχι στα public) lobbies, επιτρέπουμε την δημιουργία αυθαίρετου πλήθους άλλων private παρτίδων (αλλά όχι public, όπως εξηγείται στην συνέχεια).

Σύντομα παρουσιάστηκε ένα ακόμα μεγαλύτερο εμπόδιο σχετικά με την υλοποίηση των private lobbies. Αυτό ήταν η διαχείριση των Ids, προκειμένου το smart contract να μπορεί να υποστηρίξει πραγματικά πολλά παράλληλα παιχνίδια. Η ρίζα του προβλήματος ήταν η απαίτηση τα private παιχνίδια να χρησιμοποιούν το ίδιο struct με τα public παιχνίδια, αλλά και να παίρνουν IDs από τον ίδιο -σειριακό- μετρητή με αυτά, προκειμένου να μην χρειαστεί καμία απολύτως αλλαγή στις ήδη υλοποιημένες μεθόδους που χρησιμοποιούνται για τις δημοσιές παρτίδες. Πρακτικά δηλαδή θέλουμε να χειριζόμαστε τα ιδιωτικά παιχνίδια με τον ίδιο τρόπο όπως τα δημόσια, χρησιμοποιώντας τις ίδιες δομές δεδομένων, ώστε να μην χρειαστεί να ορίσουμε επιπλέον μεταβλητές στο storage και να κάναμε επιπλέον ελέγχους απου θα αύξαναν το κόστος.

Ποιο ήταν λοιπόν το πρόβλημα? Ας υποθέσουμε ότι αρχικά ένας απλός λογαριασμός, έστω Α, καλεί τη play() ώστε να ξεκινήσει ένα δημόσιο παιχνίδι. Καλώντας την μέθοδο, ορίζεται ως ο player1 και περιμένει έναν δεύτερο παίκτη. Το currentGameId ακόμα δεν έχει αυξηθεί καθώς θα πρέπει να αναθέσουμε τον αμέσως επόμενο παίκτη, έστω Β, που θα καλέσει την play() στο ίδιο παιχνίδι ως player2. Τι θα γίνει όμως εάν **πριν** μπει ο Player2, ένας τρίτος παίκτης (ή ακόμα και κάποιος εκ των Α ή Β) θελήσει να ξεκινήσει ένα private παιχνίδι με κάποιον άλλο? Το ιδιωτικό παιχνίδι προφανώς θα πρέπει να έχει ένα καινούργιο (και αμέσως επόμενο) ID προκειμένου να διαφοροποιηθεί από το προηγούμενο. Αν του αναθέσουμε το καινούργιο gameId και κατόπιν αυξήσουμε τον currentGameId, θα έχουμε κάνει λάθος! Το δημόσιο παιχνίδι στο οποίο περίμενε ο player1 έχει πλέον «χαθεί», καθώς ο

επόμενος παίκτης που θα καλέσει την play() θα μπει σε ένα δημόσιο παιχνίδι με καινούργιο ID ως ο Player1. Θα μπορούσαμε ενδεχομένως να ελέγχουμε κάθε φορά που ξεκινάει ένα καινούργιο private παιχνίδι, αν εκκρεμούν άλλα παιχνίδια (π.χ. διατρέχοντας τον πίνακα με τα structs των παιχνιδιών) και έτσι να χειριστούμε τα Ids κατάλληλα, ωστόσο κάτι τέτοιο θα αποτελούσε πολύ κακή σχεδιαστική επιλογή καθότι το κόστος σε gas θα ήταν απροσδιόριστο, και θα αυξανόταν όσο περισσότερα παράλληλα παιχνίδια είχαμε. Ακόμα χειρότερα, όσο ένας παίκτης περιμένει στο public lobby, μπορεί να ξεκινήσουν όσα παράλληλα παιχνίδια επιτρέπει ο μετρητής των ID (2^32 - 1). Μια τέτοια περίπτωση φαίνεται στο παρακάτω σχήμα:

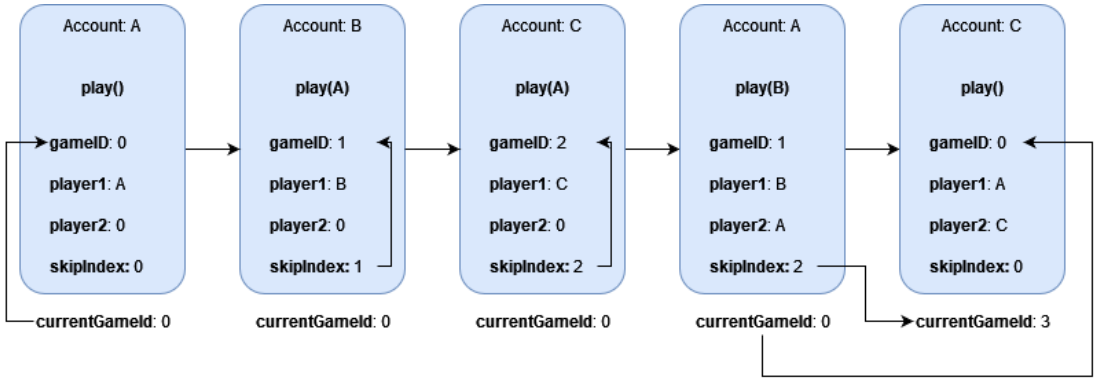


Όπως φαίνεται ανωτέρω και όπως γίνεται σαφές από την υλοποίηση των play, το πολύ ένα δημόσιο παιχνίδι μπορεί να εκκρεμεί κάθε δεδομένη στιγμή (καθώς αν καλέσει ένας ακόμα λογαριασμός την play(), το παιχνίδι ξεκινάει και περιμένουμε το επόμενο), όμως το πλήθος των ιδιωτικών παιχνιδιών που μπορεί να εκκρεμούν είναι αυθαίρετο. Το πρόγραμμα λοιπόν θα πρέπει:

a) Να «θυμάται» ποιο id είχε το δημόσιο παιχνίδι, ώστε σε μια επακόλουθη κλήση της play() να τροποποιήσει το state του σωστού παιχνιδιού και να το ξεκινήσει

b) Να παρακολουθεί και να αυξάνει τον μετρητή του ID σε κάθε κλήση της Play(address), ώστε κάθε ιδιωτικό παιχνίδι να παίρνει ένα μοναδικό αναγνωριστικό (όπως αναφέρθηκε τα ιδιωτικά παιχνίδια δεν χρειάζεται να «θυμούνται» το ids τους καθώς αυτό περιέχεται στις προσκλήσεις που στέλνει ο Player1 στον αντίπαλό του προκειμένου να ξέρει σε ποιο παιχνίδι να συμμετάσχει).

Οι παραπάνω στόχοι επιτεύχθηκαν ορίζοντας μια καθολική μεταβλητή skipIndex. Η λογική με την οποία λειτουργεί αυτή είναι η εξής: Η μεταβλητή είναι αρχικά 0, που υποδηλώνει ότι κανένα public παιχνίδι δεν εκκρεμεί (το αν εκκρεμούν ιδιωτικά παιχνίδια δεν μας απασχολεί στην υλοποίηση). Κάθε φορά που ξεκινάει ένα private lobby ενώ εκκρεμεί ένα δημόσιο, ο skipIndex παίρνει την τιμή του προηγούμενου (public) παιχνιδιού που ορίζεται μέσω του currentGameId και αυξάνει κατά ένα καθορίζοντας έτσι ID της ιδιωτικής παρτίδας. Ο καθολικός μετρητής currentGameId δεν αυξάνεται. Με αυτόν τον τρόπο, όταν κληθεί η play() ξανά, παίρνουμε τον μετρητή όπως ακριβώς τον είχαμε αφήσει κατά την αρχική κλήση και τον χρησιμοποιούμε.
Όμως, την επόμενη φορά που θα κληθεί η Play για να ξεκινήσει μια νέα δημόσια παρτίδα, αυτός ο μετρητής θα έχει μείνει πίσω κατά τόσες θέσεις όσες και τα ιδιωτικά παιχνίδια που ξεκίνησαν στον ενδιάμεσο. Για να πάρουμε λοιπόν το σωστό id, ελέγχουμε αν ο skipIndex είναι 0, κι αν όχι τον χρησιμοποιούμε για να προχωρήσουμε στο αμέσως επόμενο διαθέσιμο ID, μηδενίζοντας τον ταυτόχρονα. Αυτή η λογική φαίνεται στην ακόλουθη εικόνα:



Πλέον έχουμε πετύχει:

1. Οικονομία υλοποίησης, αφού οι επιπλέον μεταβλητές για την παρακολούθηση των states και των IDs έχουν περιοριστεί στο ελάχιστο και τα private lobbies αξιοποιούν τις ήδη υπάρχουσες δομές και μεθόδους

2. Να μπορούν να δημιουργηθούν όσα ιδιωτικά παιχνίδια θέλουν οι χρήστες μεταξύ ενός δημόσιου. Η σειρά με την οποία μπαίνουν οι χρήστες σε ένα παιχνίδι (ιδιωτικό ή δημόσιο) δεν έχει σημασία ως προς την ορθή χρήση του αναγνωριστικού

3. Δύο χρήστες να μπορούν να στείλουν και να αποδεχθούν προσκλήσεις από πολλούς άλλους, ενδεχομένως και τους ίδιους (π.χ. δύο λογαριασμοί μπορούν να ξεκινήσουν πολλά ιδιωτικά παιχνίδια μεταξύ τους)